

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**



LUẬN VĂN TỐT NGHIỆP
Ứng dụng học máy để cải thiện
Web Application Firewall

Hội đồng LVTN: Khoa học máy tính

Tập thể hướng dẫn:

TS. Nguyễn An Khương Khoa KH & KT Máy tính, ĐHBK

Nguyễn Văn Hòa Verichains Lab

Nguyễn Lê Thành VNG Corp

Giảng viên phản biện:

TS. Nguyễn Tiến Thịnh Khoa KH & KT Máy tính, ĐHBK

Sinh viên thực hiện:

Trần Ngọc Tín 1613575

Ngày 16 tháng 7 năm 2020

Lời cảm ơn

Lời nói đầu, tôi xin cảm ơn tất cả mọi người đã giúp đỡ tôi trong quá trình thực hiện luận văn. Đầu tiên, tôi xin cảm ơn thầy Nguyễn An Khương, người đã tận tình hướng dẫn cho tôi trong quá trình thực hiện đề tài cũng như định hướng rất nhiều cho tôi trong học tập và công việc. Nhờ có thầy dẫn dắt và giới thiệu, tôi đã được làm quen với vô số những người đàn anh đi trước có kinh nghiệm, nhờ đó mà kiến thức của tôi đã được mở mang rất nhiều. Thầy là một người truyền cảm hứng, một người tận tụy và nghiêm túc trong công việc, là tấm gương để tôi noi theo.

Ngoài ra, không thể không nhắc đến anh Nguyễn Văn Hòa, là một người sếp, người anh và cũng là người trực tiếp hướng dẫn cho tôi về mặt chuyên môn. Anh là một người tài năng, một người có kinh nghiệm cùng với vốn kiến thức chuyên môn vững vàng và sâu sắc, được làm việc với anh là một may mắn đối với tôi. Tôi cũng xin cảm ơn anh Nguyễn Lê Thành, anh là người đã đóng góp những ý kiến mang tính định hướng quan trọng về mặt chuyên môn cho đề tài luận văn của tôi.

Trong suốt thời gian thực hiện luận văn, tôi cảm thấy mình may mắn vì đã nhận được sự hỗ trợ tận tình của tất cả các anh/chị trong công ty Verichains. Những người đã giúp đỡ cho tôi cả về mặt chuyên môn lẫn tinh thần, tạo điều kiện thoải mái để tôi có thể tập trung hoàn thành luận văn.

Cuối cùng, tôi xin cảm ơn tất cả những người anh em trong câu lạc bộ An toàn thông tin EFIENS, mọi người đều là những người đầy tài năng và nhiệt huyết, được gặp gỡ và làm việc với mọi người là một may mắn lớn dành cho tôi. Đặc biệt, tôi xin cảm ơn anh Nguyễn Quốc Bảo, người đã tận tình hướng dẫn cho tôi về văn phong, lối viết cũng như những ý kiến đóng góp vô cùng quý báu.

Tóm tắt luận văn

Tường lửa ứng dụng web (**Web Application Firewall, WAF**), là một công nghệ bảo mật giúp giảm thiểu và ngăn chặn các tấn công thường gặp trong các website doanh nghiệp. Với thiết kế như một hàng rào chắn giữa máy chủ của doanh nghiệp và người dùng (có thể là kẻ tấn công), WAF làm nhiệm vụ phân tích, phát hiện và cảnh báo sớm những tấn công có thể xảy đến. Với mức chi phí áp dụng vừa phải, WAF là một giải pháp hoàn hảo để bảo vệ website cho các doanh nghiệp vừa và nhỏ.

Tuy nhiên, những kỹ thuật để phát hiện tấn công trên ứng dụng web hiện nay vẫn chủ yếu dựa vào phương pháp phát hiện dấu hiệu vi phạm (*signature-based detection*), hay nói cách khác là dựa trên các bộ dữ liệu tấn công có sẵn, phương pháp này có ưu điểm là đơn giản và dễ dàng áp dụng trong đa số các trường hợp. Điển hình cho phương pháp này chính là **ModSecurity WAF**. Yếu điểm chủ yếu của phương pháp này là không thể phát hiện được các vector tấn công mới cũng như phát hiện nhằm người dùng bình thường thành kẻ tấn công.

Trước những vấn đề này, mục tiêu của luận văn sẽ là phát triển một hệ thống WAF dựa vào **ModSecurity WAF** với tập luật có sẵn là **ModSecurity CRS**, trong đó tập trung khắc phục những điểm yếu hiện có của **ModSecurity CRS** dựa vào các phương pháp học máy (*machine learning*). Cụ thể, đây là một phương pháp kết hợp giữa **ModSecurity CRS** và học máy.

Mục lục

1	Giới thiệu	1
1.1	Đặt vấn đề	1
1.2	Mục tiêu	2
1.3	Thách thức	3
1.4	Giải quyết vấn đề	4
1.5	Bố cục của luận văn	4
2	Kiến thức nền tảng	6
2.1	Web application firewall	6
2.2	ModSecurity WAF	6
2.2.1	Giới thiệu	6
2.2.2	Nguyên lý hoạt động	7
2.2.3	Các chức năng cơ bản	9
2.2.4	Chu kỳ xử lý	11
2.2.5	Cú pháp luật trong ModSecurity	11
2.3	OWASP ModSecurity CRS	15
3	Thiết kế và xây dựng hệ thống	18
3.1	Giới thiệu một số công nghệ được sử dụng	18
3.1.1	Nginx và OpenResty web server	18
3.1.2	Docker container	19
3.1.3	Damn Vulnerable Web Application (DVWA)	20
3.1.4	Ngôn ngữ lập trình Lua và kỹ thuật binding với C	21
3.2	Thiết kế hệ thống	23
3.3	Xây dựng hệ thống	24
3.3.1	Cài đặt ModSecurity WAF với OpenResty Server	25
3.3.2	Cài đặt DVWA và Nginx test server	27
3.3.3	Cấu hình tên miền cho DVWA và Nginx test server	28

3.3.4	Cấu hình reverse proxy cho OpenResty server	29
3.3.5	Tích hợp ModSecurity CRS vào hệ thống	32
3.3.6	Chỉnh sửa thư viện ModSecurity	35
3.3.7	Cấu hình cơ sở dữ liệu lưu log	38
3.3.8	Cấu hình forward proxy server	42
3.4	Một số hướng tiếp cận để cải thiện hệ thống WAF	44
4	Ứng dụng học máy vào hệ thống	47
4.1	Sự kết hợp giữa ModSecurity CRS và học máy	47
4.2	Chuẩn bị dữ liệu	50
4.2.1	Vấn đề với bộ dữ liệu HTTP CSIC 2010	50
4.2.2	Sinh dữ liệu bình thường	51
4.2.3	Sinh dữ liệu tấn công	54
4.2.4	Tổng hợp kết quả	56
4.3	Huấn luyện mô hình	57
4.4	Tích hợp mô hình vào hệ thống	62
5	Xây dựng công cụ kiểm thử	66
5.1	Mục tiêu	66
5.2	Giới thiệu một số công nghệ được sử dụng	66
5.2.1	Node.js	66
5.2.2	Puppeteer	67
5.3	Chuẩn bị các testcase	67
5.4	Nguyên lý hoạt động của công cụ	68
5.4.1	Cấu trúc các module	68
5.4.2	Dấu hiệu khi WAF phát hiện tấn công	69
5.4.3	Cấu trúc của test case trong DVWA module	70
5.4.4	Định dạng của file kết quả	71
6	Kiểm định và đánh giá kết quả	73
6.1	Kiểm định mô hình học máy	73
6.2	Kiểm định với công cụ waf-test	74
7	Tổng kết	76
7.1	Các kết quả đạt được	76
7.2	Những hạn chế và hướng phát triển trong tương lai	76

Danh sách hình vẽ

2.1	Network Firewall và Web Application Firewall trong mô hình OSI	8
2.2	Nginx server làm nhiệm vụ reverse proxy giữa client và Apache server . . .	9
2.3	Ví dụ về audit log của ModSecurity	10
2.4	Ví dụ về access log trong web server Nginx	10
3.1	Kiến trúc của Docker (nguồn: www.docker.com)	20
3.2	Giao diện của ứng dụng web DVWA	21
3.3	Kiến trúc hệ thống WAF	24
3.4	Trang chủ sau khi cài đặt thành công OpenResty server	27
3.5	Cơ chế chuyển tiếp request từ client thông qua OpenResty server	29
3.6	HTTP Request tới http://dvwa.test	31
3.7	Kết quả sau khi truy cập đường dẫn http://dvwa.test	31
3.8	Kết quả sau khi truy cập đường dẫn http://nginx.test	32
3.9	Tấn công vào ứng dụng web DVWA	35
3.10	Giao diện ứng dụng Mongo Express sau khi cài đặt thành công	40
3.11	Ứng dụng web DVWA bị lộ cơ sở dữ liệu bởi tấn công SQL injection . . .	46
4.1	Chuyển tiếp traffic trên cổng 80/443 thông qua proxy server	52
4.2	Giao diện của tiện ích Proxy SwitchyOmega	53
4.3	Công cụ sinh dữ liệu tấn công	56
4.4	Cấu trúc của dữ liệu trước khi huấn luyện với học máy	58
4.5	Mô hình cây Decision Tree sau khi huấn luyện	60
5.1	Tấn công bị WAF phát hiện và ngăn chặn	70
5.2	Chạy công cụ waf-test với ứng dụng DVWA	72
5.3	Kết quả kiểm thử với tấn công SQL Injection trên DVWA	72
6.1	Ma trận tương quan của mô hình Decision Tree	74
6.2	Ma trận tương quan của mô hình Random Forest	74

Danh sách bảng

2.1	Danh sách các chỉ thị luật trong ModSecurity	12
2.2	Danh sách các biến thông dụng trong ModSecurity	13
2.3	Danh sách các toán tử thông dụng trong ModSecurity	14
2.4	Danh sách các toán tử thông dụng trong ModSecurity	15
4.1	Danh sách 15 tên miền có số lượng request nhiều nhất	54
4.2	Thống kê số lượng trong tập dữ liệu	57
4.3	Tập dữ liệu được sử dụng trong mô hình	57
6.1	Kết quả kiểm định với công cụ <code>waf-test</code>	75

Danh sách thuật ngữ viết tắt

WAF	Web Application Firewall
ML	Machine Learning
OWASP	Open Web Application Security Project
CRS	Core Rule Set
DDoS	Distributed Denial-of-Service
FP	False Positive
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
DVWA	Damn Vulnerable Web Application
JS	Javascript
CNN	Convolutional neural network
XSS	Cross-Site Scripting
JSON	Javascript Object Notation
URI	Uniform Resource Identifier
API	Application Programming Interface

1.1 Đặt vấn đề

Với sự phát triển chóng mặt của công nghệ đi kèm theo đó là sự tinh vi, đa dạng trong các loại hình tấn công của tội phạm không gian mạng hiện nay, bảo mật thông tin nói chung và bảo mật ứng dụng web nói riêng mang một vai trò hết sức quan trọng và cấp thiết. Nhằm chống lại những nguy cơ bị tấn công cũng như bị đánh cắp dữ liệu, các tổ chức, doanh nghiệp,... cần có sự quan tâm và đầu tư đúng mức về độ bảo mật trong các sản phẩm của mình, đặc biệt là các ứng dụng web. Tuy nhiên, để đầu tư một đội ngũ gồm các chuyên gia bảo mật nhằm kiểm tra và vá lỗi liên tục trong các sản phẩm của mình, một doanh nghiệp cần phải có một sự đầu tư tài chính tương đối cao và không phải doanh nghiệp nào cũng có thể chi trả được, đặc biệt là các doanh nghiệp vừa và nhỏ. Hai giải pháp khá phổ biến hiện nay mà các doanh nghiệp vừa và nhỏ có thể áp dụng riêng biệt hoặc đồng thời.

- Nhằm đảm bảo an toàn cho sản phẩm trước khi vận hành, mã nguồn ứng dụng của doanh nghiệp nên được kiểm tra bởi các công ty bên thứ ba chuyên về bảo mật. Ngoài ra, doanh nghiệp có thể tham gia vào các chương trình bug bounty ¹ như HackerOne ², nơi các sản phẩm của doanh nghiệp được kiểm tra bởi các chuyên gia bảo mật làm việc độc lập với cơ chế trao thưởng tùy thuộc theo độ nguy hiểm và phức tạp của lỗ hổng.
- Trong quá trình vận hành, mặc dù đã được kiểm tra trước đó, nhưng những lỗ hổng bảo mật vẫn có thể còn sót lại do nhiều yếu tố khác nhau. Có thể là do đội ngũ kiểm thử bảo mật không tìm thấy, cũng có thể do ứng dụng đã được cập nhật lên phiên bản và xuất hiện thêm những lỗ hổng bảo mật mới chưa được kiểm tra lại. Nhằm tăng cường bảo vệ cho ứng dụng web, các giải pháp trong quá trình vận hành như rà soát, phân tích hành vi người truy cập được thực hiện độc lập với quá trình

¹Bug bounty: chương trình cho phép các hacker bên ngoài tham gia tìm kiếm lỗ hổng bảo mật, mỗi lỗ hổng bảo mật sẽ được trả tiền dựa vào mức độ nguy hiểm và mức độ lan rộng

²HackerOne: nền tảng bug bounty phổ biến nhất hiện nay, <https://hackerone.com>

kiểm thử trước đó. Trong đó, các ứng dụng web sẽ được đặt dưới sự bảo vệ của một lớp tường lửa gọi là **Web Application Firewall (WAF)** ³. Đây cũng là nội dung nghiên cứu chính của đề tài.

WAF về cơ bản là một lớp tường lửa đứng giữa website và người dùng, trong đó có các kẻ tấn công (*hacker*). Khi muốn tấn công vào website của một doanh nghiệp, các hacker phải tìm cách vượt qua được sự phát hiện và ngăn chặn của tường lửa. WAF có thể giúp vá tạm thời những lỗ hổng sẵn có trong mã nguồn của ứng dụng website hay còn gọi là vá ảo (*virtual patching*) ⁴, cảnh báo những tấn công thất bại và đặc biệt hiệu quả trong việc phòng chống các cuộc tấn công DDoS ⁵. Tuy không thể giúp chặn được hoàn toàn những tấn công khai thác các lỗ hổng bảo mật sẵn có trong ứng dụng nhưng WAF có thể giúp tạm thời ngăn chặn những người dùng có nguy cơ tấn công và gây hại tới hệ thống. Tuy nhiên các công nghệ WAF hiện tại vẫn còn mang những điểm yếu nhất định như phát hiện nhầm người dùng bình thường thành hacker, chưa thể phát hiện được một số tấn công tinh vi, phức tạp và điển hình là những *zero-day attack*. *Zero-day attack* tức là những tấn công nhằm vào các lỗ hổng bảo mật chỉ mới vừa phát hiện ra trên hệ thống và vẫn chưa có bản vá lỗi.

Trong phạm vi của đề tài, chúng tôi sẽ tập trung phân tích, đánh giá điểm mạnh yếu của những phương pháp phổ biến được ứng dụng trong các công nghệ WAF hiện nay. Đồng thời đề xuất một số phương pháp để cải tiến độ hiệu quả và hiện thực một hệ thống WAF hoàn chỉnh.

1.2 Mục tiêu

Đề tài này được thực hiện nhằm phát triển và cải thiện công nghệ WAF thông dụng hàng đầu hiện nay là **ModSecurity**, thông qua sự kết hợp giữa phương pháp phát hiện dấu hiệu vi phạm (*signature-based*) hay nói cách khác là những tập luật cố định được tạo ra từ những vectơ tấn công có sẵn và sự linh hoạt của các phương pháp học máy. Cụ thể hơn là sự kết hợp giữa **ModSecurity Core Rule Set (CRS)** và các giải thuật học máy.

Mục tiêu cuối cùng của đề tài là nhằm cải tiến độ chính xác của **ModSecurity WAF**, đồng thời vẫn duy trì được hiệu năng cao cho hệ thống. Về chi tiết, cần đạt được một số yêu cầu sau.

- Phát hiện và ngăn chặn kịp thời những tấn công nhằm khai thác một số lỗ hổng phổ biến được liệt kê trong **OWASP Top 10** [8]. Đây là tập hợp 10 lỗi bảo mật phổ

³Web application firewall: tường lửa bảo vệ ứng dụng web

⁴Virtual patching: kỹ thuật vá tạm thời các lỗ hổng bảo mật mà không cần sửa trực tiếp vào mã nguồn ứng dụng

⁵DDoS: distributed denial of service (tấn công từ chối dịch vụ)

biến nhất trong các ứng dụng web được công bố bởi tổ chức OWASP⁶.

- Giảm thiểu độ dương tính giả (*false positive*, FP). Thông thường các công nghệ WAF dựa trên dấu hiệu vi phạm (*signature-based*) hay nói cách khác là các tập luật có sẵn như ModSecurity CRS thường có độ FP khá cao, chính vì vậy mục tiêu của đề tài cần thực hiện là giảm độ FP xuống mức tối thiểu, đây cũng là điểm quan trọng mấu chốt của WAF.
- Duy trì tốc độ xử lý của WAF ở mức ổn định. Nếu một hệ thống WAF phải trải qua quá nhiều bước xử lý tính toán phức tạp cho mỗi HTTP request sẽ dẫn tới việc làm tăng thời gian xử lý và bị giới hạn về số lượng người dùng request cùng lúc, hậu quả là làm cho hệ thống bị quá tải và có thể gây ảnh hưởng lớn đến người dùng.
- Hệ thống WAF cần dễ dàng tích hợp vào các ứng dụng web có sẵn. Để đạt được yêu cầu này, việc ứng dụng các công nghệ container như Docker⁷ là một yêu cầu thiết yếu.

1.3 Thách thức

Trong quá trình nghiên cứu và thực hiện đề tài luận văn, chúng tôi nhận ra một số điểm khó khăn cần phải giải quyết như sau.

Về vấn đề dữ liệu. Bộ dữ liệu ban đầu mà chúng tôi định sử dụng cho đề tài luận văn là HTTP CSIC 2010 [2]. Tuy nhiên trong quá trình phân tích và tìm hiểu bộ dữ liệu này, chúng tôi đã nhận ra một số điểm hạn chế và có phần ảnh hưởng lớn đến kết quả của việc huấn luyện mô hình học máy. Chi tiết về những hạn chế của bộ dữ liệu này sẽ được đề cập trong Chương 4.

Về vấn đề tích hợp. Việc áp dụng các mô hình học máy vào Nginx là một vấn đề không hề đơn giản. Đa phần việc huấn luyện cũng như trích xuất các mô hình học máy hiện nay đều được thực hiện bởi các thư viện của ngôn ngữ lập trình Python, điển hình là Scikit-Learn, TensorFlow, PyTorch,... Ngoài trừ các mô hình đơn giản như Linear Regression hay Linear Classification, việc áp dụng các mô hình học máy từ ngôn ngữ Python vào Nginx đều cần phải trải qua các bước chuyển đổi phức tạp. Một cách tiếp cận đơn giản là fork một process để gọi mã nguồn Python và thực thi mô hình học máy, sau đó gửi trả kết quả về cho Nginx. Một cách tiếp cận đơn giản khác có thể là giao tiếp với mô hình học máy thông qua web API. Tuy nhiên, cả hai hướng tiếp cận này đều gây giảm sút hiệu năng của hệ thống WAF một cách đáng kể. Có thể nói, trong trường hợp này, hệ thống sẽ xảy ra hiện tượng nghẽn cổ chai (*bottleneck*). Phân tích chi tiết về điểm yếu của hai hướng tiếp cận này sẽ được đề cập trong Chương 4. Trong thực tế, hai

⁶OWASP: tổ chức phi lợi nhuận, mã nguồn mở, nghiên cứu chuyên sâu về bảo mật nhằm đưa ra các tiêu chuẩn về an toàn phần mềm cho cộng đồng <https://owasp.org/>

⁷Docker: công nghệ container phổ biến nhất hiện nay, giúp cho việc triển khai, phát triển các dự án phần mềm trở nên dễ dàng hơn. <https://www.docker.com/why-docker>

hướng tiếp cận này chỉ phù hợp với các hệ thống nhỏ và mang tính chất minh họa, không thể áp dụng cho một website thực tế với lưu lượng truy cập cao cùng số lượng người dùng lớn. Đặc biệt, nếu WAF được dùng để bảo vệ nhiều ứng dụng web cùng lúc thì những giải pháp trên sẽ hoàn toàn không khả thi.

1.4 Giải quyết vấn đề

Để giải quyết các khó khăn trên, chúng tôi đã tiến hành tìm hiểu nhiều hướng khác nhau và hiện tại đã tìm được một số giải pháp.

Về vấn đề dữ liệu. Với mục đích là bổ sung, cập nhật HTTP traffic mới, phù hợp để ứng dụng thực tế, chúng tôi đã tiến hành dựng lên một hệ thống proxy và điều hướng toàn bộ HTTP traffic trong môi trường cục bộ đến hệ thống WAF. Cụ thể, chúng tôi đã chuyển hướng toàn bộ lưu lượng truy cập web hàng ngày của mình đến hệ thống WAF được cấu hình trước đó. Mục tiêu là dùng chính traffic của mình để huấn luyện cho mô hình học máy. Điều này có thể giải quyết được vấn đề traffic chỉ tập trung vào một website. Theo hướng này, vấn đề dữ liệu bị lệch theo hướng chủ quan vẫn không thể tránh khỏi. Tuy nhiên, với dữ liệu sinh ra theo hướng này vẫn mang tính chất tổng quát và thực tế hơn so với bộ dữ liệu ban đầu là HTTP CSIC 2010. Ngoài ra, mô hình học máy sau khi huấn luyện có thể dùng được một cách ổn định cho một số trang web thông dụng. Đối với traffic tấn công, bộ kiểm thử đơn vị (*unit test*) của ModSecurity CRS là FTW⁸ sẽ được sử dụng để sinh dữ liệu tự động kết hợp với một số công cụ tấn công khác.

Về vấn đề tích hợp. Trong quá trình nghiên cứu, chúng tôi đã tìm ra được một số thư viện cho phép để chuyển đổi mô hình học máy từ Python sang ngôn ngữ khác nhằm tích hợp vào Nginx, trong đó có thư viện Scikit-Learn-Porter⁹ dùng để chuyển đổi các mô hình từ Scikit-Learn qua các ngôn ngữ thông dụng khác như Java, Javascript, C,.. Hiện tại thư viện này vẫn còn đang được phát triển và vẫn còn nhiều giới hạn về số lượng ngôn ngữ cũng như mô hình hỗ trợ. Sau khi đã chuyển mô hình sang mã nguồn C, chúng tôi sẽ tiến hành biên dịch và tích hợp trực tiếp vào Nginx. Chi tiết về cách tích hợp sẽ được đề cập trong Chương 4.

1.5 Bố cục của luận văn

Bố cục của luận văn được chia làm 7 chương, sơ lược nội dung của từng chương như sau.

⁸FTW: công cụ kiểm thử tự động dành cho các hệ thống WAF được viết bằng ngôn ngữ Python, tham khảo mã nguồn tại <https://github.com/CRS-support/ftw>

⁹Scikit-Learn-Porter: mã nguồn của thư viện có thể tham khảo tại <https://github.com/nok/sklearn-porter>

Chương 1 - Giới thiệu

Trình bày tổng quan về vấn đề cần giải quyết, mục tiêu của đề tài. Nêu sơ lược những khó khăn gặp phải và hướng giải quyết.

Chương 2 - Kiến thức nền tảng

Giới thiệu một số kiến thức, công nghệ nền tảng được ứng dụng trong đề tài luận văn.

Chương 3 - Thiết kế và xây dựng hệ thống

Đề xuất kiến trúc và xây dựng hệ thống WAF, chuẩn bị cho việc tích hợp mô hình học máy.

Chương 4 - Ứng dụng học máy vào hệ thống

Phân tích, mô tả quá trình huấn luyện và ứng dụng mô hình học máy vào WAF.

Chương 5 - Xây dựng công cụ kiểm thử

Quá trình chuẩn bị các bộ kiểm thử và hiện thực công cụ kiểm thử.

Chương 6 - Kiểm định và đánh giá kết quả

Mô tả, tiến hành các phương pháp kiểm định đối với WAF.

Chương 7 - Tổng kết

Tổng kết công việc đã đạt được, những hạn chế và hướng phát triển trong tương lai.

2.1 Web application firewall

Web application firewall (WAF) là một tường lửa tầng ứng dụng, nhằm bảo vệ các ứng dụng giao tiếp dựa trên giao thức HTTP. WAF sẽ đặt ra những tập quy tắc (rule set) thông qua HTTP request và response giữa người dùng với server. Những tập quy tắc này thông thường dùng để bảo vệ các ứng dụng web bên phía server khỏi một số loại tấn công cơ bản chẳng hạn như SQL injection, XSS (Cross site scripting), Command injection,... Thông thường một proxy server sẽ tập trung bảo vệ cho người dùng (chủ yếu là bảo vệ danh tính). Ngược lại, WAF làm nhiệm vụ bảo vệ các server, WAF có thể bảo vệ một hoặc nhiều các ứng dụng web khác nhau, ta có thể coi WAF như một reverse proxy ¹.

2.2 ModSecurity WAF

2.2.1 Giới thiệu

ModSecurity là một công cụ Web application firewall (WAF) mã nguồn mở, đa nền tảng được xây dựng và phát triển bởi SpiderLabs của Trustwave [12]. Công cụ này cung cấp một cú pháp luật đơn giản và mạnh mẽ, cho phép dễ dàng viết các luật nhằm bảo vệ các ứng dụng web khỏi nhiều loại tấn công khác nhau, đồng thời cho phép kiểm soát, ghi vào tệp nhật ký (*log file*) và phân tích thời gian thực đối với HTTP traffic. Với trên 10000 ứng dụng sử dụng trên toàn thế giới, ModSecurity có thể nói là ứng dụng WAF được dùng nhiều nhất ở thời điểm hiện tại. [14]

ModSecurity là một module độc lập có thể được cài đặt trong các web server phổ biến như Apache, Nginx, IIS,... Các phiên bản trước 3.0 là một module phụ thuộc vào web server Apache, để cài đặt được trên những nền tảng khác chẳng hạn như Nginx, ta cần phải cài đặt một số module cần thiết của web server Apache. Nhưng kể từ phiên bản 3.0

¹Reverse proxy: là một server đứng phía trước, nhận HTTP request thay cho các server đứng sau

trở lên, ModSecurity đã được viết lại thành một module mới với tên gọi **LibModSecurity**, việc module hóa khiến cho ModSecurity không còn phụ thuộc vào nền tảng web server Apache như các phiên bản trước và sẽ dễ dàng tích hợp vào bất kỳ hệ thống nào. Để cài đặt LibModSecurity vào các nền tảng web server khác nhau ta cần các module *connector*. Các module connector có thể xem như là lớp tiếp hợp (*interface*) cho phép kết nối giữa web server đến thư viện LibModSecurity.

2.2.2 Nguyên lý hoạt động

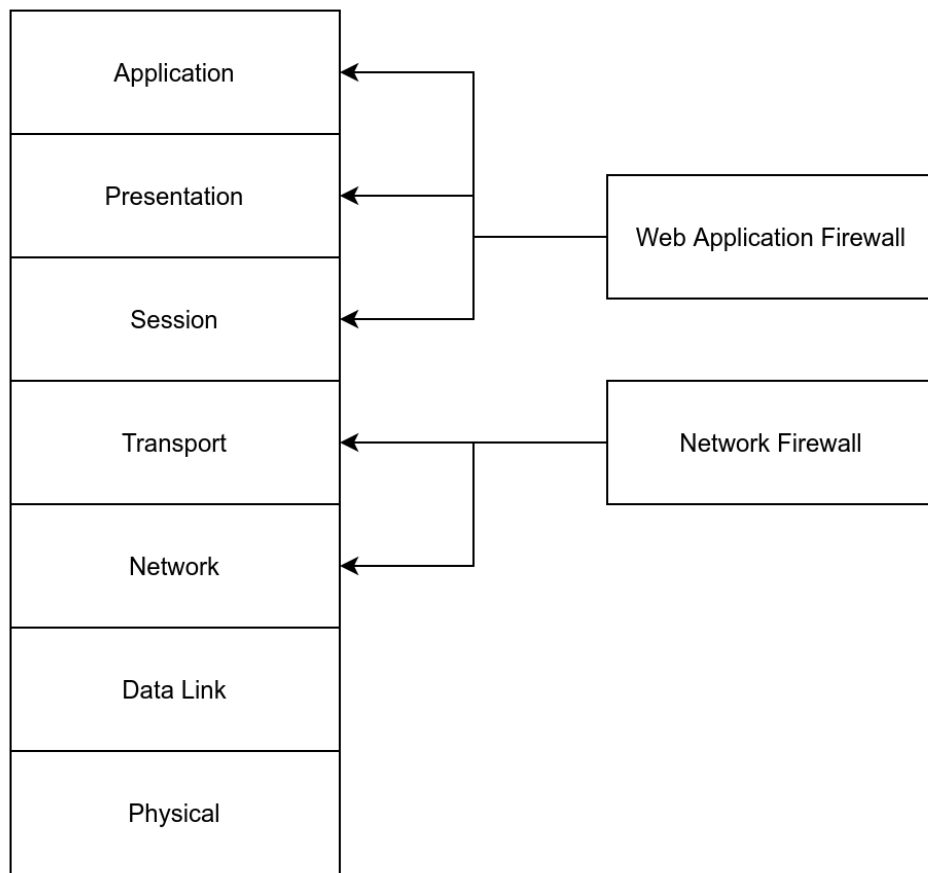
Khác với những hệ thống tường lửa hoạt động ở tầng mạng (network firewall), những hệ thống tường lửa này thông thường không thể can thiệp hay phân tích nội dung của request bởi hầu hết các web server hiện nay đều hỗ trợ giao thức HTTPS và hầu hết đều sử dụng giao thức này. Chính vì điều này, việc ngăn chặn được những request độc hại trở thành bất khả thi đối với các loại tường lửa này bởi các HTTP request đã bị mã hóa thông qua SSL/TLS². Trái ngược lại, ModSecurity sẽ bắt đầu phân tích các request sau khi những request này đã được giải mã hoàn toàn, điều này giúp chúng ta có thể viết những tập luật dựa vào nội dung của HTTP request để biết được liệu đây có phải là request tấn công hay không. Hình 2.1 so sánh vị trí của 2 loại tường lửa này trong mô hình OSI³

Lấy một ví dụ với ModSecurity được cài đặt tích hợp vào server Nginx đóng vai trò reverse proxy, những HTTP request sẽ thông qua Nginx server trước khi đến với server thực sự cần được bảo vệ, giả sử server này dùng công nghệ Apache. Các bước xử lý HTTPS request giữa người dùng với Apache server có thể được mô tả như sau:

- Nginx server giải mã SSL.
- Nginx server đọc các HTTP request từ TCP stream đã được giải mã.
- Nginx server phân tích nội dung (parse) các HTTP request.
- Nginx gửi các HTTP request qua ModSecurity để phân tích.
- ModSecurity phân tích và xử lý request dựa trên những tập luật được cấu hình sẵn và sẽ ngăn chặn hoặc cảnh báo bất kỳ lúc nào nếu phát hiện, nghi ngờ tấn công từ người dùng.
- Sau khi phân tích và xác định request này là an toàn bởi ModSecurity, Nginx gửi request đó đến server Apache để xử lý.
- Server Apache sau khi xử lý xong request sẽ trả về kết quả cho server Nginx.

²SSL/TLS: là một giao thức tầng ứng dụng giúp mã hóa dữ liệu trong quá trình truyền tải giữa các máy trong mạng máy tính

³Mô hình OSI (Open Systems Interconnection): mô hình tham chiếu kết nối các hệ thống mở - là một thiết kế dựa vào nguyên lý tầng cấp, lý giải một cách trừu tượng kỹ thuật kết nối truyền thông giữa các máy vi tính và thiết kế giao thức mạng giữa chúng



Hình 2.1: Network Firewall và Web Application Firewall trong mô hình OSI

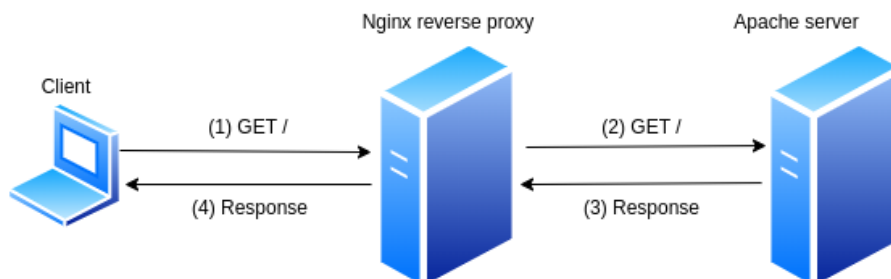
- Nginx nhận kết quả từ Apache và phân tích tiếp bằng ModSecurity để xác định lần nữa xem có phải là bị tấn công hay không. Tùy theo cài đặt của ModSecurity, ta có thể chặn kết quả đến người dùng nếu như phát hiện server Apache xuất hiện lỗi khi gặp những **status code 5xx** trả về (Đây là những status code ⁴ thông báo Apache server gặp lỗi, những status code này có thể đưa ra nhiều thông tin giúp cho những kẻ tấn công dễ dàng khai thác như phiên bản Apache đang được sử dụng, tệ hơn nữa có thể bị lộ những thông báo **Exception error** ⁵ mang nhiều dữ liệu nhạy cảm) hoặc gặp những từ khóa không mong muốn trong **response body** ⁶.
- Sau khi kiểm tra và hoàn toàn không phát hiện lỗi, lúc này Nginx server mới mã hóa HTTP response và gửi trả về kết quả cho người dùng. Toàn bộ quá trình có thể

⁴**Status code**: là mã trạng thái của HTTP request trả về, cho phép biết được request thành công hoặc có lỗi xảy ra

⁵**Exception error**: là một dạng tin nhắn thông báo của hệ thống khi có lỗi xảy ra, dựa vào những thông báo này, các nhà phát triển có thể dễ dàng biết được nguyên nhân cũng như cách sửa lỗi trong hệ thống

⁶**Response body**: chứa nội dung trả về của HTTP request

được minh họa như hình 2.2.



Hình 2.2: Nginx server làm nhiệm vụ reverse proxy giữa client và Apache server

Nhận xét: Người dùng sẽ hoàn toàn không thể biết được IP thật của server Apache thông qua mô hình này, mọi HTTP request phải được thông qua reverse proxy là Nginx. Mô hình này hiện tại đang được áp dụng bởi hầu hết doanh nghiệp cung cấp giải pháp về WAF như Imperva, CloudFlare, Sucuri, Cloudrity, Polaris,... Ngoài ra cần chú ý rằng giao tiếp giữa Nginx với Apache server không nhất thiết phải là HTTP mà có thể là HTTPS.

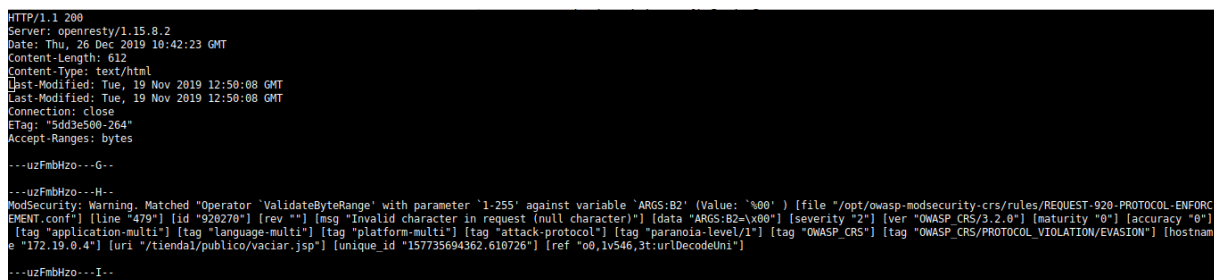
2.2.3 Các chức năng cơ bản

Parsing. Sau khi các HTTP request được phân tích bởi web server, toàn bộ HTTP request sẽ được gửi qua ModSecurity để tiến hành xử lý. Điều này dẫn đến việc phân tích HTTP request sẽ bị lặp lại, tuy nhiên quá trình này cũng không tốn quá nhiều tài nguyên và có thể chấp nhận được. Kết quả của quá trình phân tích HTTP request sẽ được dùng trong việc phân tích bởi Rule Engine.

Buffering. Trong cài đặt mặc định, toàn bộ request và response body của HTTP request sẽ được lưu vào bộ nhớ đệm hay còn gọi là “buffering”. Buffering giúp cho ModSecurity có thể phân tích được toàn bộ request trước khi gửi đến ứng dụng web được yêu cầu và phân tích toàn bộ response trước khi gửi kết quả lại cho người dùng. Việc này giúp ModSecurity phát hiện được dấu hiệu tấn công trong request gửi đến hoặc sự rò rỉ thông tin nhạy cảm dựa trên response trả về thông qua Rule Engine (sẽ đề cập trong phần tiếp theo). Quá trình buffering có thể khiến cho server proxy phải tốn nhiều RAM hơn bình thường để lưu trữ HTTP request/response. Do đó, để đảm bảo hiệu năng của hệ thống, các server proxy thường bỏ qua bước kiểm tra request/response body trong thực tế (chỉ giữ lại HTTP header).

Logging. Logging cũng là một chức năng quan trọng nữa mà ModSecurity cung cấp, logging trong ModSecurity là *audit log*. Audit log cho phép ghi lại toàn bộ quá trình gửi nhận request của client và server cuối bao gồm cả request/response header và body như trong hình 2.3. Cần lưu ý, *audit log* khác với *access log* thông thường trong web server,

access log trong web server chỉ cho chúng ta biết một số thông tin cơ bản như URI, phiên bản HTTP, status code, User-Agent, địa chỉ IP,... Ví dụ về access log có thể xem ở hình 2.4. Cần chú ý, nếu log toàn bộ những request đi tới web server sẽ làm giảm hiệu năng của server đáng kể do phải thực hiện rất nhiều những tác vụ IO bound, may mắn là ModSecurity cung cấp cho chúng ta một số cấu hình hữu ích như chỉ ghi log những request bị nghi ngờ, hoặc có thể chỉ định điều kiện để ghi vào log trong quá trình thực thi bởi Rule Engine. Ví dụ như, khi gặp bất kỳ request nào có cụm từ `<script>` hoặc chứa endpoint `/admin/login.php` thì sẽ ghi vào audit log.



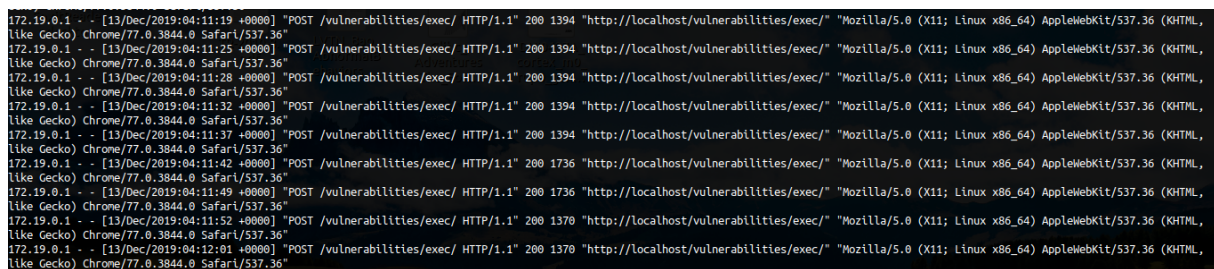
```

HTTP/1.1 200
Server: openresty/1.15.8.2
Date: Thu, 26 Dec 2019 10:42:23 GMT
Content-Length: 612
Content-Type: text/html
Last-Modified: Tue, 19 Nov 2019 12:50:08 GMT
Etag: "5dd3e500-264"
Accept-Ranges: bytes

...-uzFmbHzo--G--
ModSecurity: Warning: Matched "Operator 'ValidateByteRange' with parameter '1-255' against variable 'ARGS:B2' (Value: '%00' ) [file "/opt/owasp-modsecurity-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "479"] [id "920270"] [rev ""] [msg "Invalid character in request (null character)"] [data "ARGS:B2=x00"] [severity "2"] [ver "OWASP_CRS/3.2.0"] [maturity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "OWASP_CRS/PROTOCOL_VIOLATION/EVASION"] [hostname "172.19.0.4"] [uri "/tiendal/publico/vaciar.jsp"] [unique_id "157735694362.610726"] [ref "o0,1v546,3t:urlDecodeUni"]
...-uzFmbHzo--I--

```

Hình 2.3: Ví dụ về audit log của ModSecurity



```

172.19.0.1 - - [13/Dec/2019:04:11:19 +0000] "POST /vulnerabilities/exec/ HTTP/1.1" 200 1394 "http://localhost/vulnerabilities/exec/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3844.0 Safari/537.36"
172.19.0.1 - - [13/Dec/2019:04:11:25 +0000] "POST /vulnerabilities/exec/ HTTP/1.1" 200 1394 "http://localhost/vulnerabilities/exec/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3844.0 Safari/537.36"
172.19.0.1 - - [13/Dec/2019:04:11:28 +0000] "POST /vulnerabilities/exec/ HTTP/1.1" 200 1394 "http://localhost/vulnerabilities/exec/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3844.0 Safari/537.36"
172.19.0.1 - - [13/Dec/2019:04:11:32 +0000] "POST /vulnerabilities/exec/ HTTP/1.1" 200 1394 "http://localhost/vulnerabilities/exec/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3844.0 Safari/537.36"
172.19.0.1 - - [13/Dec/2019:04:11:37 +0000] "POST /vulnerabilities/exec/ HTTP/1.1" 200 1394 "http://localhost/vulnerabilities/exec/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3844.0 Safari/537.36"
172.19.0.1 - - [13/Dec/2019:04:11:42 +0000] "POST /vulnerabilities/exec/ HTTP/1.1" 200 1376 "http://localhost/vulnerabilities/exec/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3844.0 Safari/537.36"
172.19.0.1 - - [13/Dec/2019:04:11:49 +0000] "POST /vulnerabilities/exec/ HTTP/1.1" 200 1376 "http://localhost/vulnerabilities/exec/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3844.0 Safari/537.36"
172.19.0.1 - - [13/Dec/2019:04:11:52 +0000] "POST /vulnerabilities/exec/ HTTP/1.1" 200 1370 "http://localhost/vulnerabilities/exec/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3844.0 Safari/537.36"
172.19.0.1 - - [13/Dec/2019:04:12:01 +0000] "POST /vulnerabilities/exec/ HTTP/1.1" 200 1370 "http://localhost/vulnerabilities/exec/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3844.0 Safari/537.36"

```

Hình 2.4: Ví dụ về access log trong web server Nginx

Rule Engine. Rule Engine là thành phần quan trọng nhất của ModSecurity WAF, nhiệm vụ của Rule Engine là phân tích các HTTP request/response sau khi đã qua giai đoạn parsing. Rule Engine cho phép người dùng viết các luật bảo vệ theo cú pháp được quy định bởi ModSecurity, những luật này sẽ được thực thi trên mỗi HTTP request/response nhằm kiểm tra xem có phải web server đang bị tấn công hay không. Những luật này thông thường là các biểu thức chính quy (regular expression) được dùng để kiểm tra trên các trường như request parameters, request body,... hoặc cũng có thể gọi các đoạn mã viết bằng Lua bên ngoài để thực hiện những phân tích phức tạp hơn. Dựa vào kết quả phân tích, Rule Engine có thể đưa ra cảnh báo hoặc thực thi một số hành động nhằm chống lại việc tấn công cũng như ghi lại HTTP request vào audit log nếu cần thiết.

2.2.4 Chu kỳ xử lý

Ở phần trước, chúng ta đã đề cập đến một số chức năng của ModSecurity, ở phần này sẽ tập trung phân tích về chu kỳ xử lý các HTTP request của ModSecurity. Chu kỳ ở đây tính từ lúc người dùng gửi HTTP request và kết thúc khi nhận được response từ server proxy. Trong ModSecurity, một chu kỳ sẽ bao gồm 5 giai đoạn (5 phase), các luật của ModSecurity có thể được thực hiện ở bất kỳ giai đoạn nào tùy theo cấu hình của mỗi luật, kết quả của giai đoạn này có thể ảnh hưởng tới giai đoạn tiếp theo hoặc không.

Giai đoạn 1 - Request Header

“Request Header” theo như tên gọi là giai đoạn sau khi ModSecurity nhận được toàn bộ request header nhưng chưa tới request body, giá trị của các request header cho ta biết được cách để xử lý request body. Chẳng hạn ta có thể viết một luật khi gặp header `Content-type: application/json` ta có thể biết cách đọc request body theo dạng tương ứng là JSON.

Giai đoạn 2 - Request Body

Giai đoạn “Request Body” diễn ra ngay sau khi nhận được toàn bộ HTTP request body. Ta có thể viết luật phân tích request body tại giai đoạn này.

Giai đoạn 3 - Response Header

Giai đoạn “Response Header” diễn ra ngay sau khi nhận được toàn bộ HTTP response header nhưng trước khi nhận được response body. Những luật trong giai đoạn này thường là để quyết định xem có nên phân tích hay bỏ qua response body. Ví dụ như nếu phát hiện ra content-type là css thì có thể không cần kiểm tra body nữa.

Giai đoạn 4 - Response Body

Giai đoạn “Response body” là bước kiểm tra cuối cùng nếu cần thiết, các luật ở bước này có thể dùng để kiểm tra liệu có lỗi xuất hiện (*Exception*) trong nội dung trả về hay không.

Giai đoạn 5 - Logging

“Logging” là giai đoạn duy nhất mà ModSecurity không thể ra những quyết định như chặn người dùng. Giai đoạn này chủ yếu được dùng để kiểm soát được việc ghi log diễn ra như thế nào.

2.2.5 Cú pháp luật trong ModSecurity

Trong ngôn ngữ viết luật của ModSecurity có tổng cộng 9 chỉ thị [9], danh sách các chỉ thị có thể được tham khảo trong bảng 2.1.

Bảng 2.1: Danh sách các chỉ thị luật trong ModSecurity

Chỉ thị	Mô tả
SecAction	Thực hiện hành động không có điều kiện, nghĩa là 1 luật luôn luôn xảy ra
SecDefaultAction	Chỉ định các hành động mặc định, những luật theo sau khi xảy ra sẽ thực thi
SecMarker	Các luật khi tạo marker sẽ không thực hiện hành động mà chỉ gán ID
SecRule	Tạo luật mới
SecRuleInheritance	Kiểm tra xem luật có được kế thừa trong ngữ cảnh con
SecRuleRemoveById	Xóa luật với ID tương ứng
SecRuleRemoveByMsg	Xóa luật có message trùng với biểu thức chính quy (regular expression)
SecRuleScript	Tạo ra luật dùng mã Lua bên ngoài
SecRuleUpdateActionById	Thay các hành động của luật ứng với ID bằng danh sách các hành động mới

ModSecurity cung cấp khá nhiều chỉ thị giúp tạo ra các luật bảo vệ (*security rules*). Nhưng khi viết các luật bổ sung, ta chỉ cần chú ý vào chỉ thị **SecRule**, đây là chỉ thị được sử dụng nhiều nhất khi tạo luật. Trong phạm vi đề tài, nhóm chỉ xin tóm tắt sơ lược những kiến thức cần thiết đủ để hiểu về cách ModSecurity hoạt động, để có được cái nhìn chi tiết hơn có thể tham khảo trong tài liệu ModSecurity Handbook [9].

Mỗi luật bảo vệ được viết với chỉ thị SecRule sẽ có dạng chuẩn như sau:

```
SecRule VARIABLES "OPERATOR" ["TRANSFORMATIONS,ACTIONS"]
```

Trong cú pháp viết luật (*rule syntax*) ở trên có tổng cộng 4 thành phần, trong đó có 2 thành phần bắt buộc là biến (*variable*), toán tử (*operator*) và 2 thành phần không bắt buộc là hàm biến đổi (*transformation*) và hành động (*action*). Nếu trong một luật không có 2 thành phần transformation hoặc action thì ModSecurity sẽ ngầm định dùng các transformations, actions trong chỉ thị **SecDefaultAction**.

Biến

Biến (variable) xác định trường nào của HTTP transaction sẽ được dùng để tính toán. Sau quá trình parsing thì chi tiết của HTTP transaction được lưu vào các biến để các luật bảo vệ thực hiện kiểm tra lên đó. Các biến sẽ ở đây sẽ lưu giá trị ở dạng chuỗi nhị phân (binary string), do đó việc tính toán, so sánh sẽ an toàn ngay cả khi chuỗi chỉ gồm các ký tự không in được. Trong mỗi luật phải có ít nhất một biến. Bảng 2.2 là danh sách một số biến thông dụng.

Bảng 2.2: Danh sách các biến thông dụng trong ModSecurity

Biến	Mô tả
ARGS	Danh sách các tham số request
ARGS_NAMES	Danh sách tên các tham số request
ARGS_POST	Danh sách tham số trong request body
FILES	Danh sách file trong request
FILES_NAMES	Danh sách tên file trong request
REQUEST_URI	Request URI
REQUEST_HEADERS	Danh sách request header
REQUEST_METHOD	Phương thức request (GET, POST, ...)
REQUEST_FILENAME	Lấy đường dẫn từ request URI (không tính tham số)
REQUEST_BODY	Request body
RESPONSE_BODY	Response body
TX	Biến tạm dùng để lưu các giá trị cho nhiều luật
XML	Biến giúp truy xuất XML DOM

Toán tử

Toán tử (operator) xác định cách mà biến (sau khi thực hiện các hàm chuyển đổi) sẽ được tính toán. ModSecurity hỗ trợ khá nhiều toán tử thông dụng, nhưng trong đó toán tử quan trọng nhất và thường được sử dụng nhất trong các luật là so sánh biểu thức chính quy (regular expression). Ngoài ra ModSecurity còn cho phép bổ sung toán tử mới, mỗi luật chỉ cho phép dùng một toán tử. Bảng 2.3 là danh sách một số toán tử thông dụng trong ModSecurity.

Bảng 2.3: Danh sách các toán tử thông dụng trong ModSecurity

Toán tử	Mô tả
@rx	So sánh biểu thức chính quy (regular expression)
@contains	Kiểm tra chuỗi con
@streq	So sánh 2 chuỗi bằng nhau
@eq	So sánh 2 số bằng nhau
@ge	So sánh 2 số lớn hơn hoặc bằng
@gt	So sánh 2 số lớn hơn
@le	So sánh 2 số bé hơn hoặc bằng
@lt	So sánh 2 số bé hơn
@detectXSS	Kiểm tra XSS injection
@detectSQLi	Kiểm tra SQL injection

Hàm chuyển đổi

Nhiều hàm chuyển đổi (transformation function) có thể được thiết lập trong một luật để thực hiện tiền xử lý các biến (variable) trước khi luật bắt đầu phân tích dựa trên toán tử. Các hàm chuyển đổi thường được dùng để giải mã giá trị của biến bị mã hóa trước khi tính toán hoặc ngược lại. Bảng 2.4 là danh sách một số hàm chuyển đổi thông dụng.

Hành động

Trong ngữ cảnh hiện tại, hành động (action) hiểu theo cách đơn giản là các công việc cần được thực thi khi một request/response vi phạm vào luật tương ứng, mỗi luật có thể có nhiều hoặc không có hành động nào.

Một số luật ví dụ

Ví dụ 1:

```
SecRule REQUEST_URI "@rx <script>" "log,deny,status:403"
```

Giải thích: luật này có nghĩa là nếu trong request URI có chứa chuỗi <script> thì ghi vào log sau đó chặn người dùng và trả về status code 403. Trong đó, toán tử @rx dùng để kiểm tra chuỗi theo dạng biểu thức chính quy (regular expression).

Ví dụ 2:

Bảng 2.4: Danh sách các toán tử thông dụng trong ModSecurity

Hàm chuyển đổi	Mô tả
<code>none</code>	Reset hàm chuyển đổi
<code>lowercase</code>	Chuyển đổi ký tự thành chữ thường
<code>base64Decode</code>	Giải mã base64
<code>normalizePath</code>	Chuẩn hóa cho đường dẫn
<code>normalizePathWin</code>	Chuẩn hóa cho đường dẫn trong Window
<code>urlDecode</code>	Giải mã URL encoding
<code>urlDecodeUni</code>	Giải mã URL encoding (dùng cho Microsoft %u)
<code>removeNulls</code>	Xóa tất cả null byte trong đầu vào
<code>htmlEntityDecode</code>	Giải mã cho HTML entity encoding
<code>length</code>	Tính độ dài của biến
<code>compressWhitespace</code>	Xóa các khoảng trắng dư thừa

```
SecRule ARGS "@contains delete from" \
    phase:2,t:lowercase,t:compressWhitespace,block
```

Giải thích: luật này trước hết sẽ biến đổi tất cả request parameters thành chữ thường (lowercase) và xóa những khoảng trắng dư thừa, sau đó kiểm tra xem nếu có request parameter nào chứa chuỗi `delete from` thì sẽ ngăn chặn người dùng gửi request đó. Cụm `phase:2` ở đây là metadata action dùng để báo hiệu xem luật này thực thi ở giai đoạn nào (nếu đọc giá trị biến ở sai giai đoạn sẽ dẫn đến giá trị biến rỗng). Cần chú ý, ở đây các cụm như `t:lowercase` và `t:compressWhitespace` là các hàm chuyển đổi.

Lưu ý: Trên đây chỉ là một số cú pháp luật cơ bản của ModSecurity nhằm mục đích giới thiệu sơ lược cách ModSecurity hoạt động, ngoài những luật đơn giản như trên, ModSecurity còn hỗ trợ những cú pháp phức tạp hơn như luật liên kết (chain rule) và lệnh nhảy (skip rule). Với cú pháp mà ModSecurity cung cấp, người sử dụng có thể dễ dàng bổ sung các luật cụ thể cho từng ứng dụng khác nhau.

2.3 OWASP ModSecurity CRS

Trong phần trước, chúng tôi đã trình bày một số cú pháp cơ bản trong khi viết luật bảo vệ với ModSecurity WAF. Hiện tại, chỉ với ModSecurity thôi thì chưa đủ để xây dựng một hệ thống tường lửa có thể dùng được, ta cần phải bổ sung các tập luật giúp ngăn

chặn web server khỏi bị tấn công. Tuy nhiên, việc xây dựng lại từ đầu các tập luật giúp chống được những loại tấn công phổ biến là điều không dễ dàng và dễ gây ra sai sót. Ở đây chúng tôi sẽ sử dụng OWASP ModSecurity Core Rule Set, bao gồm những luật thiết yếu giúp web server chống lại được nhiều loại tấn công khác nhau, được xây dựng và phát triển bởi nhiều chuyên gia bảo mật web uy tín thuộc tổ chức OWASP.

OWASP ModSecurity CRS là một tập những luật tổng quát được sử dụng trong ModSecurity WAF [3]. Mục tiêu của OWASP ModSecurity CRS nhằm giúp bảo vệ ứng dụng web khỏi nhiều loại tấn công khác nhau, bao gồm OWASP top 10 [8], với độ sai lệch tối thiểu.

- SQL Injection (SQLi)
- Cross Site Scripting (XSS)
- Local File Inclusion (LFI)
- Remote File Inclusion (RFI)
- Remote Code Execution (RCE)
- PHP Code Injection
- HTTP Protocol Violations
- HTTPoxy
- Shellshock
- Session Fixation
- Scanner Detection
- Metadata/Error Leakages
- Project Honey Pot Blacklist
- GeoIP Country Blocking

Nguyên lý hoạt động của ModSecurity CRS khá đơn giản, khi một request/response được kiểm tra, mỗi luật được cấu hình trong CRS sẽ sinh ra tương ứng một số điểm cho request/response gọi là “điểm bất thường” (anomaly score). Khi tổng điểm này vượt một ngưỡng điểm cho phép thì request của người dùng sẽ bị chặn. Ngoài ra, chúng ta có thể điều chỉnh mức độ ngăn chặn (paranoia level) của ModSecurity CRS. Trong ModSecurity CRS có 4 mức độ ngăn chặn, có thể mô tả sơ lược như sau.

- **Mức 1.** Đây là mức độ mặc định, chỉ bật những luật cơ bản nhất nhằm giảm độ FP xuống mức tối thiểu.

- **Mức 2.** Mức này bổ sung thêm một số luật cho SQL, XSS và code injection. Ngoài ra chống được một số tấn công nâng cao, đây là mức độ bảo mật trung bình.
- **Mức 3.** Mức độ này tập trung vào những tấn công ít thông dụng hơn và cũng chống được các dạng vượt mặt (bypass) WAF. Đây là mức độ dành cho những người có yêu cầu mức độ bảo mật cao và có kinh nghiệm xử lý độ FP thông qua các luật bổ sung.
- **Mức 4.** Đây là mức độ với độ bảo mật rất cao, có thể giới hạn những ký tự đặc biệt. Mức độ này cũng sẽ dẫn đến độ FP tăng rất cao và cần được xử lý riêng biệt, rất hiếm ứng dụng nào sử dụng mức này.

Thiết kế và xây dựng hệ thống

3.1 Giới thiệu một số công nghệ được sử dụng

3.1.1 Nginx và OpenResty web server

Nginx là một ứng dụng mã nguồn mở, có thể được dùng như web server, reverse proxy, load balancer, media streamer,... Nổi bật bởi hiệu năng và tính ổn định cao, đây là một giải pháp hàng đầu được áp dụng bởi nhiều doanh nghiệp. Nginx được thiết kế theo mô hình hướng sự kiện, xử lý bất đồng bộ, mã nguồn được viết hoàn toàn bằng C. Có thể nói, Nginx là một trong những web server có hiệu năng tốt nhất hiện nay.

OpenResty là một phiên bản mở rộng của Nginx, dựa trên Nginx core, cho phép sử dụng ngôn ngữ lập trình Lua để cấu hình và viết các module mở rộng. Ngoài ra, các module mở rộng được cung cấp bởi OpenResty giúp cho các nhà phát triển có thể dễ dàng tùy biến Nginx theo nhiều cách khác nhau, đồng thời vẫn giữ được hiệu năng cao. Với cú pháp đơn giản của ngôn ngữ lập trình Lua cũng như khả năng tương tác linh hoạt với các thư viện viết bằng C, OpenResty đã giúp cải tiến Nginx trở thành một nền tảng web server vô cùng mạnh mẽ. Bên dưới là một file cấu hình đơn giản của OpenResty server, trong đó web server sẽ mở kết nối trên giao thức HTTP ở cổng 80 và trả về người dùng một trang web với nội dung “hello, world” khi được yêu cầu.

```
worker_processes 1;
error_log logs/error.log;
events {
    worker_connections 1024;
}

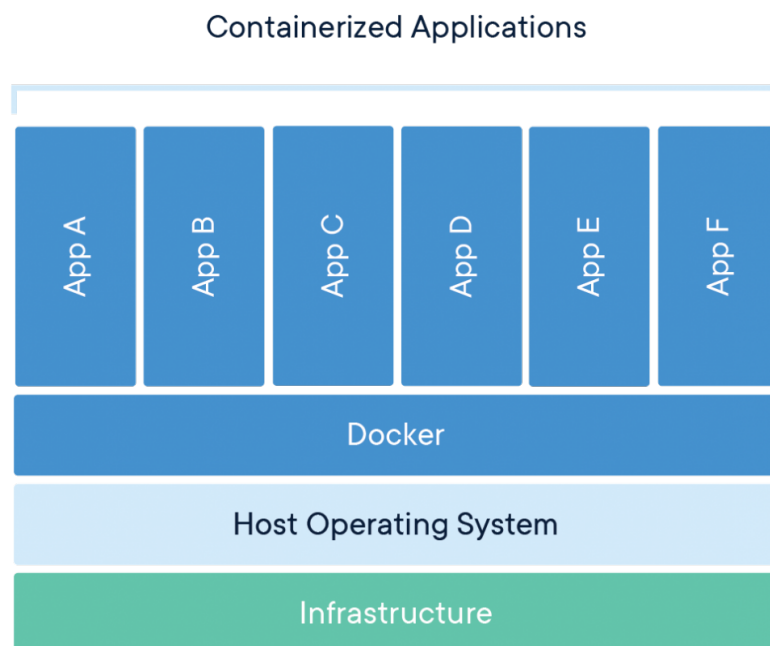
http {
    server {
        listen 80;
        location / {
            default_type text/html;
            content_by_lua_block {
                ngx.say("<p>hello, world</p>")
            }
        }
    }
}
```

3.1.2 Docker container

Docker container là một đơn vị phần mềm tiêu chuẩn, đóng gói toàn bộ mã nguồn và dữ liệu cần thiết để dễ dàng chạy trên nhiều môi trường khác nhau. Docker container image là một gói phần mềm, trong đó bao gồm toàn bộ mọi thứ cần thiết có thể để chạy một ứng dụng bao gồm: mã nguồn, môi trường, công cụ, thư viện hệ thống và cấu hình. Khi một docker container image được khởi động, nó sẽ trở thành docker container (tương tự như file thực thi và process). Công nghệ Docker giúp cho các ứng dụng dựa trên nền hệ điều hành Linux có thể chạy được trên môi trường Windows hoặc MacOS. Do đó, Docker đóng một vai trò quan trọng trong công việc cài đặt, triển khai hệ thống cũng như hỗ trợ mạnh mẽ cho quá trình phát triển ứng dụng.

Điểm mạnh của Docker container là cung cấp môi trường hoàn toàn độc lập cho mỗi ứng dụng tương tự như máy ảo, do đó giúp dễ dàng xử lý vấn đề không tương thích môi trường giữa nhiều ứng dụng cùng chạy trên một máy chủ. Tuy cung cấp môi trường độc lập nhưng Docker container có dung lượng nhẹ hơn và có tốc độ xử lý tốt nhiều so với máy ảo thông thường bởi cơ chế dùng chung phần lõi của hệ điều hành (kernel) giữa máy chủ và container, trong khi máy ảo phải giả lập lại toàn bộ kernel của hệ điều hành (xem hình 3.1). Đối với máy chủ chạy hệ điều hành Windows hoặc MacOS thì vẫn cần phải giả lập Linux kernel khi sử dụng Docker, do đó Docker chỉ phát huy được hiệu năng tối đa khi chạy trên hệ điều hành Linux.

Việc ứng dụng công nghệ Docker sẽ giúp cho quá trình xây dựng và cài đặt hệ thống WAF trở nên dễ dàng hơn. Cụ thể, hệ thống WAF sẽ chia ra thành các module nhỏ theo kiến trúc micro service. Trong đó, mỗi Docker container sẽ là một service độc lập và có một chức năng cụ thể. Để dễ dàng quản lý cũng như giao tiếp giữa các Docker container, chúng tôi sử dụng Docker Compose. Với công cụ này, chúng tôi có thể dễ dàng định nghĩa, khởi tạo các container và giả lập một mạng máy tính cá nhân (private network) kết nối

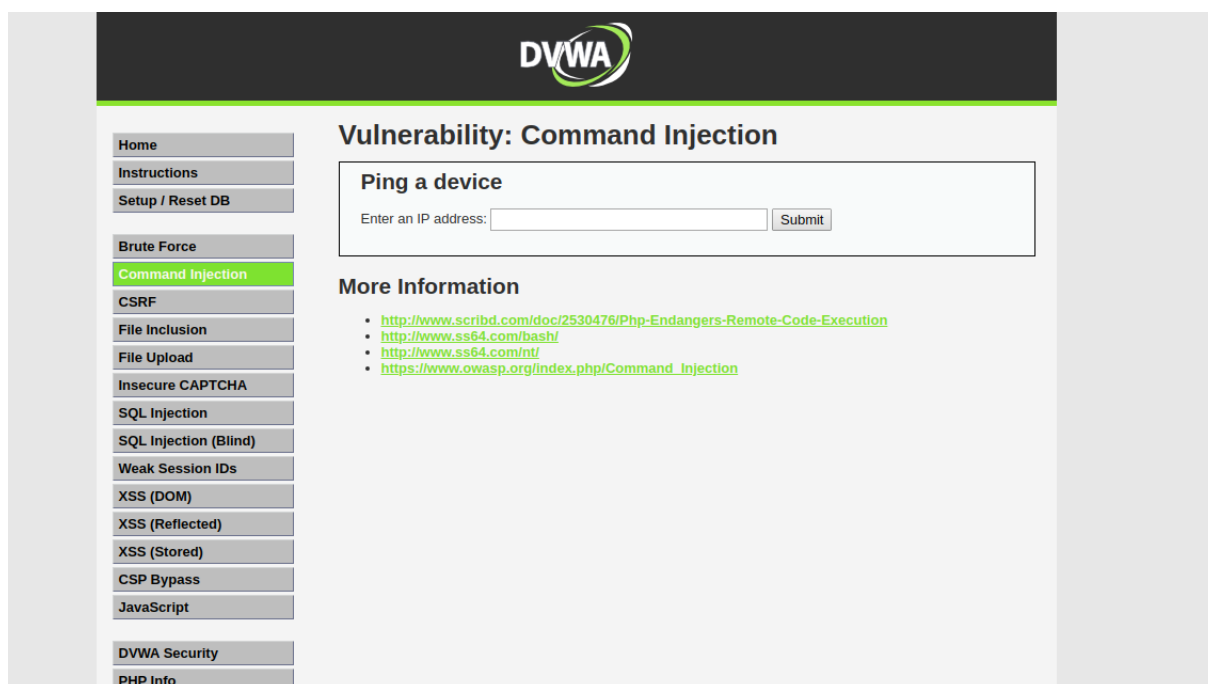


Hình 3.1: Kiến trúc của Docker (nguồn: www.docker.com)

các container với nhau. Sau khi thiết kế và xây dựng hệ thống trên môi trường cục bộ, chúng tôi có thể dễ dàng triển khai hệ thống lên máy chủ trên môi trường điện toán đám mây nhờ vào công nghệ Docker.

3.1.3 Damn Vulnerable Web Application (DVWA)

Damn Vulnerable Web App (DVWA) [11] là một ứng dụng web được viết trên nền tảng PHP/MySQL với vô số các lỗ hổng bảo mật khác nhau. Mục tiêu của DVWA là cung cấp một môi trường nghiên cứu, thử nghiệm các lỗ hổng bảo mật một cách hợp pháp cho những chuyên gia về an toàn thông tin. Ngoài ra, DVWA cũng là một môi trường thử nghiệm, dạy học hiệu quả về bảo mật cho các trường học và doanh nghiệp. Trong quá trình xây dựng môi trường kiểm thử cho WAF, chúng tôi đã chọn ứng dụng web DVWA như một phần của quá trình kiểm thử. Hình 3.2 là giao diện chính của DVWA sau khi cài đặt và đăng nhập thành công.



Hình 3.2: Giao diện của ứng dụng web DVWA

3.1.4 Ngôn ngữ lập trình Lua và kỹ thuật binding với C

Lua là một ngôn ngữ lập trình mã nguồn mở với đặc tính gọn nhẹ, mạnh mẽ và có khả năng nhúng vào nhiều ứng dụng viết bằng các ngôn ngữ khác. Với cú pháp đơn giản, cho phép khai báo biến kiểu động, chạy và xử lý byte code trên máy ảo theo hướng register-based, hỗ trợ quản lý bộ nhớ tự động, đây là một ngôn ngữ phù hợp cho cấu hình và viết kịch bản (scripting). Một số ứng dụng của Lua có thể kể đến như OpenResty server, trong đó sử dụng ngôn ngữ Lua để viết file cấu hình, cho phép các lập trình viên tùy biến và viết các module riêng khá dễ dàng so với phiên bản gốc là Nginx. Ngoài ra, Lua còn có thể được sử dụng để viết các luật bổ sung trong ModSecurity, chi tiết sẽ được đề cập ở Chương 4.

Điểm mạnh nhất của ngôn ngữ lập trình Lua, đó là khả năng nhúng vào các ứng dụng khác. Một chương trình có thể đăng ký các hàm (function) cần thiết vào môi trường của Lua, các hàm này có thể được hiện thực bằng C, C++ hoặc nhiều ngôn ngữ khác. Do đó, khi viết các đoạn mã bằng Lua và nhúng vào ứng dụng, người dùng có thể sử dụng các module, thư viện có sẵn của Lua và cả những module được viết bằng C/C++. Những ưu điểm trên khiến cho Lua trở thành một ngôn ngữ có tính mở rộng cao. Các hàm trong thư viện được viết bằng C có thể được gọi trực tiếp từ mã nguồn Lua. Ngược lại, các hàm trong thư viện Lua có thể được gọi trực tiếp từ mã nguồn C thông qua Lua C API.

Chúng ta sẽ xem xét một ví dụ cơ bản về cách viết một thư viện C và đăng ký dưới dạng Lua module để có thể gọi trực tiếp từ mã nguồn Lua. Trong module này, chúng ta

sẽ đăng ký hàm `mysin()` là một wrapper của hàm `sin()` trong C.

```
static int my_sin (lua_State *L) {
    double arg = luaL_checknumber(L, 1);
    lua_pushnumber(L, sin(arg));
    return 1;
}

static const struct luaL_Reg mylib [] = {
    {"mysin", my_sin},
    {NULL, NULL}
};

// register functions
// we must name this function luaopen_mylib (important)
int luaopen_mylib (lua_State *L) {
    luaL_newlib(L, mylib);
    return 1;
}
```

Để sử dụng được module trên trong Lua, chúng ta sẽ tiến hành biên dịch đoạn mã trên thành một module với tên là `mylib`. Đoạn mã nguồn bên dưới minh họa cách sử dụng module `mylib` từ Lua.

```
local mylib = require 'mylib'

-- calling C sin() function using mysin()
print(mylib.mysin(2))
```

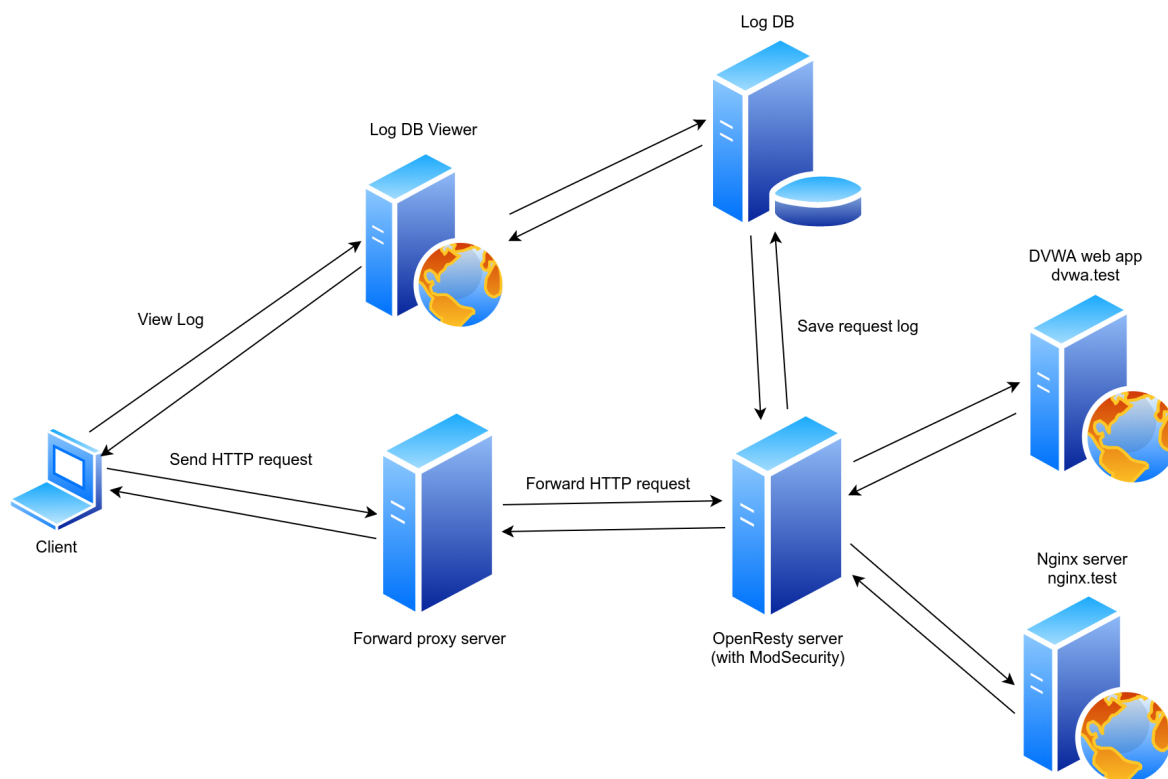
Ngoài cách đăng ký các hàm trong C thông qua Lua C API như trên, chúng ta còn có thể gọi hàm C trực tiếp từ Lua thông qua thư viện Lua FFI ¹.

¹Lua FFI: thư viện cho phép gọi trực tiếp các hàm C từ Lua mà không cần viết các thủ tục binding như cách thông thường. Tham khảo thêm tại https://luajit.org/ext_ffi.html

3.2 Thiết kế hệ thống

So với quá trình thiết kế ban đầu trong giai đoạn đề cương luận văn, kiến trúc hiện tại của hệ thống sẽ có một vài thay đổi. Về cơ bản, chúng tôi sẽ tiến hành bổ sung thêm một số service như cơ sở dữ liệu chứa log (*log db*), ứng dụng truy vấn cơ sở dữ liệu log (*log db viewer*) và một proxy server làm nhiệm vụ chuyển tiếp. Kiến trúc của hệ thống hiện tại có thể được mô tả như trong hình 3.3, hệ thống sẽ được chia làm nhiều service nhỏ theo kiến trúc micro service, trong đó mỗi service chỉ làm một nhiệm vụ cụ thể, công dụng của mỗi service có thể mô tả ngắn gọn như sau.

- Server OpenResty với vai trò là WAF, mọi traffic HTTP sẽ được điều hướng đến để phân tích. Đây là thành phần quan trọng nhất của hệ thống, server này sẽ được cài đặt ModSecurity cùng với ModSecurity CRS. Mô hình học máy sau khi huấn luyện cũng sẽ được tích hợp vào.
- Server cài đặt ứng dụng web DVWA, phục vụ cho quá trình kiểm thử khả năng phòng thủ của WAF trong các giai đoạn sau.
- Server bổ sung, được cài đặt sẵn web server Nginx, dùng trong quá trình sinh dữ liệu tấn công cũng như quá trình chuyển đổi bộ dữ liệu HTTP CSIC 2010 sang định dạng cần thiết nhằm phục vụ quá trình huấn luyện mô hình học máy.
- Server MongoDB làm nhiệm vụ chứa request log được gửi đến từ WAF. Cần lưu ý, dữ liệu được gửi đến không phải là dữ liệu thô (HTTP request), toàn bộ dữ liệu đều được chuẩn hóa theo một định dạng do chúng tôi định nghĩa trước khi lưu vào cơ sở dữ liệu.
- Server cài đặt ứng dụng MongoExpress. Đây là một ứng dụng web với giao diện trực quan, giúp dễ dàng quản lý và thao tác với cơ sở dữ liệu MongoDB.
- Server còn lại sẽ được cài đặt Nginx và cấu hình để thực hiện nhiệm vụ là forward proxy server. Server này sẽ được sử dụng cho quá trình thu thập dữ liệu bình thường trong máy tính cá nhân cũng như quá trình kiểm thử thực tế sau khi tích hợp mô hình học máy vào WAF.



Hình 3.3: Kiến trúc hệ thống WAF

3.3 Xây dựng hệ thống

Sau quá trình thử nghiệm, hiện tại trong giai đoạn này, chúng tôi đã đề xuất ra một kiến trúc tương đối hoàn chỉnh cho hệ thống WAF. Dựa vào kiến trúc hệ thống đã đề xuất, quá trình xây dựng hệ thống sẽ bao gồm một số bước sau.

- Cài đặt ModSecurity WAF với OpenResty Server
- Cài đặt DVWA và Nginx test server
- Cấu hình tên miền cho DVWA và Nginx test server
- Cấu hình reverse proxy cho OpenResty server
- Tích hợp ModSecurity CRS vào hệ thống
- Chỉnh sửa thư viện ModSecurity
- Cấu hình cơ sở dữ liệu lưu log
- Cấu hình forward proxy server

3.3.1 Cài đặt ModSecurity WAF với OpenResty Server

Trước hết, ta sẽ tiến hành cấu hình và cài đặt OpenResty Server. Toàn bộ mã nguồn của hệ thống cũng như hướng dẫn cài đặt có thể tham khảo trên trang github của chúng tôi tại <https://github.com/ngoctint11vc/waf>.

Chúng tôi sẽ tiến hành cài đặt OpenResty server phiên bản 1.15.8 trong một docker container. Việc trước tiên, ta cần biên dịch thư viện ModSecurity từ mã nguồn, mã nguồn của ModSecurity có thể tải về tại <https://github.com/SpiderLabs/ModSecurity>. Như đã giới thiệu ở phần trước, kể từ phiên bản 3.0 trở đi, ModSecurity đã được viết thành một thư viện riêng và không còn phụ thuộc vào các module của web server Apache. Dưới đây là một ví dụ đơn giản về cách dùng thư viện ModSecurity được tham khảo từ trang github chính thức của ModSecurity [12].

```
#include "modsecurity/modsecurity.h"
#include "modsecurity/transaction.h"

char main_rule_uri[] = "basic_rules.conf";

int main (int argc, char **argv)
{
    ModSecurity *modsec = NULL;
    Transaction *transaction = NULL;
    Rules *rules = NULL;

    modsec = msc_init();

    rules = msc_create_rules_set();
    msc_rules_add_file(rules, main_rule_uri);

    transaction = msc_new_transaction(modsec, rules);

    msc_process_connection(transaction, "127.0.0.1");
    msc_process_uri(transaction, "http://www.modsecurity.org/test?key1=
value1&key2=value2&key3=value3&test=args&test=test");
    msc_process_request_headers(transaction);
    msc_process_request_body(transaction);
    msc_process_response_headers(transaction);
    msc_process_response_body(transaction);

    return 0;
}
```

Ví dụ trên chỉ mang tính chất tham khảo, trong quá trình cấu hình cũng như cài đặt ModSecurity WAF, chúng ta sẽ dùng 1 module khác là ModSecurity-nginx [13]. Module này sẽ đóng vai trò là connector, cung cấp một tầng trung gian để giao tiếp giữa OpenResty server với ModSecurity. Cần lưu ý, do OpenResty vẫn dựa trên Nginx core nên các

module của Nginx hoàn toàn có thể sử dụng được cho OpenResty. Sau khi tải mã nguồn của module ModSecurity-Nginx và OpenResty về, chúng ta sẽ biên dịch OpenResty server với module ModSecurity-Nginx và cài đặt bằng các lệnh như bên dưới.

```
#!/bin/bash

cd openresty-1.15.8.2
./configure -j4 \
    --with-pcre-jit \
    --with-ipv6 \
    --add-module=/opt/ModSecurity-nginx
make -j4
make install
```

Trong ví dụ trên, chúng ta giả định rằng openresty-1.15.8.2 là thư mục hiện tại chứa mã nguồn của OpenResty server và /opt/ModSecurity-nginx chứa mã nguồn của module connector ModSecurity-Nginx.

Sau khi đã cài đặt xong OpenResty server, bước tiếp theo ta cần vào file cấu hình của Nginx trong OpenResty tại đường dẫn /usr/local/openresty/nginx/conf/nginx.conf và điều chỉnh một số cấu hình căn bản như bên dưới.

```
http {
    server {
        listen      80;
        server_name localhost;

        location / {
            root html;
        }

        # redirect server error pages to the static page /50x.html
        error_page   500 502 503 504   /50x.html;
        location = /50x.html {
            root      html;
        }
    }
}
```

Ta tiến hành kiểm tra lại quá trình cài đặt OpenResty server bằng cách vào đường dẫn `http://localhost` trong trình duyệt. Nếu cấu hình chính xác, chúng ta sẽ thấy trang giới thiệu của OpenResty như hình 3.4.

Chú ý: Các bước cài đặt trên đây chỉ tóm tắt một số phần quan trọng, cấu hình đầy đủ có thể tham khảo trong Dockerfile tại <https://github.com/ngoactint11vc/waf/blob/master/openresty/Dockerfile>.

Welcome to OpenResty!

If you see this page, the OpenResty web platform is successfully installed and working. Further configuration is required.

For online documentation and support please refer to openresty.org.
Commercial support is available at openresty.com.

Thank you for flying OpenResty.

Hình 3.4: Trang chủ sau khi cài đặt thành công OpenResty server

3.3.2 Cài đặt DVWA và Nginx test server

Bước tiếp theo, ta sẽ tiến hành cài đặt hai server DVWA và Nginx server nhằm tạo môi trường kiểm thử cho WAF. Với công nghệ Docker container, các bước cài đặt tương đối đơn giản. Ta chỉ cần tải 2 docker image từ trang hub.docker.com² là DVWA tại <https://hub.docker.com/r/vulnerables/web-dvwa/> và Nginx tại https://hub.docker.com/_/nginx. Sau khi tải 2 image về máy, ta khởi tạo và chạy hai docker container với tên dvwa-test và nginx-test sử dụng file cấu hình docker-compose.yml như bên dưới.

```
version: '3.7'
services:
  openresty:
    build:
      context: ./openresty
      dockerfile: Dockerfile
    container_name: openresty
    ports:
      - "80:80"
      - "443:443"
    volumes:
      ...
  dvwa-test:
    image: vulnerables/web-dvwa
    container_name: dvwa-test
  nginx-test:
    image: nginx:1.17.6
    container_name: nginx-test
    ...
```

²hub.docker.com: đây là trang web chia sẻ các Docker container image có sẵn, được đóng góp bởi cộng đồng sử dụng Docker

Chú ý: sau khi cài đặt xong DVWA và Nginx test server như trên, chúng ta vẫn chưa thể truy cập được vào hai server vừa mới cài đặt, lý do là chúng ta đã dành hai cổng 80 (HTTP) và 443 (HTTPS) cho server OpenResty. Để kiểm tra được việc cài đặt cho hai server trên ta cần thực hiện bước tiếp theo.

3.3.3 Cấu hình tên miền cho DVWA và Nginx test server

Để truy cập được 2 server DVWA và Nginx vừa mới cài đặt, chúng ta phải tiến hành cấu hình tên miền (domain) cho 2 server này. Ý tưởng cơ bản là ta sẽ trỏ tên miền của cả 2 server này về IP của server OpenResty. Tiếp theo, chúng ta sẽ bàn về hai cách cấu hình trong hai môi trường là production và development.

Production. Trong môi trường production, server OpenResty sẽ được cài đặt riêng biệt trong 1 máy chủ và có địa chỉ IP Public³ cố định. Giả định rằng hai server DVWA và Nginx sẽ có domain lần lượt là `dvwa.com` và `nginx.com`, lúc này ta sẽ thêm hai A record trong mục quản lý DNS với nội dung như bên dưới.

```
dvwa.com    IN  A    [openresty server IP]
nginx.com   IN  A    [openresty server IP]
```

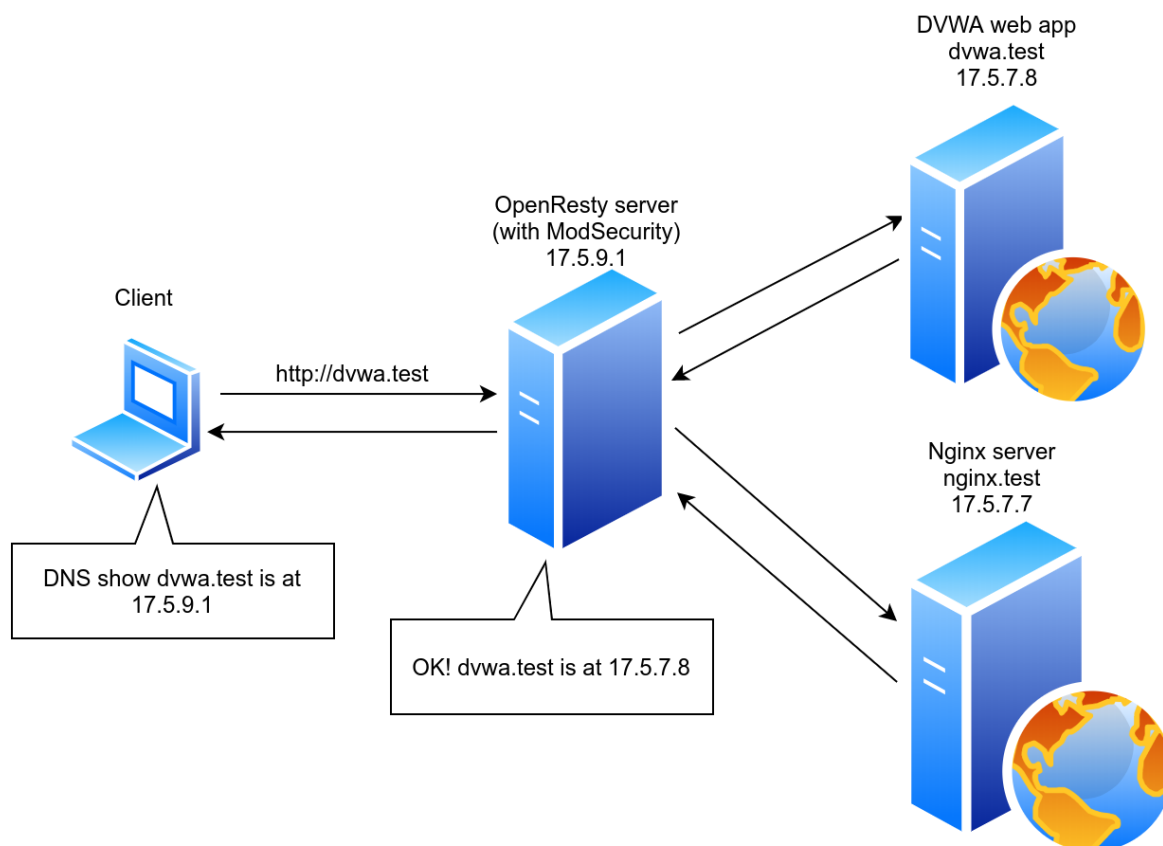
Sau khi thêm hai record trên, mọi HTTP request tới server `dvwa.com` hoặc `nginx.com` sẽ được gửi đến OpenResty server. Câu hỏi đặt ra là làm thế nào để OpenResty server nhận biết được request nào đến `dvwa.com` và request nào đến `nginx.com`? Câu trả lời đơn giản là dựa vào Host header trong HTTP request gửi đến, nếu Host header có nội dung `dvwa.com` thì OpenResty sẽ chuyển tiếp request tới IP thực sự của server DVWA (còn gọi là *origin server IP*), các tên miền khác hoàn toàn tương tự. Trong server OpenResty sẽ cần phải có cơ chế nào đó để lưu trữ việc “mapping” giữa tên miền và IP thật của server cần chuyển request đến, có thể lưu trong file hoặc cơ sở dữ liệu tùy ý. Quá trình chuyển tiếp request từ client tới server tương ứng có thể được mô tả như trong hình 3.5.

Development. Trong môi trường development, chúng ta sẽ tận dụng công nghệ Docker để tạo ra một mạng máy tính cá nhân. Trước hết, ta tiến hành đặt tên miền cho hai server DVWA và Nginx lần lượt là `dvwa.test` và `nginx.test` (có thể chọn tên miền tùy ý). Trong môi trường của hệ điều hành Linux, ta có thể dễ dàng can thiệp vào quá trình phân giải tên miền (resolve domain) trong máy bằng cách thêm vào file `/etc/hosts` các dòng bên dưới.

```
127.0.0.1    dvwa.test
127.0.0.1    nginx.test
```

Lúc này, mọi request đến `dvwa.test` hoặc `nginx.test` sẽ được điều hướng về IP loop-back 127.0.0.1 tức là địa chỉ IP của máy tính hiện tại. Cần lưu ý, trong file cấu hình

³IP Public: là địa chỉ IP do nhà cung cấp dịch vụ Internet cấp phát, địa chỉ IP Public là định danh cần thiết để mọi người trên Internet có thể truy cập được vào server



Hình 3.5: Cơ chế chuyển tiếp request từ client thông qua OpenResty server

`docker-compose.yml` chúng ta đã thực hiện “mapping” 2 cổng 80 và 443 trong máy tính bên ngoài vào bên trong container tương ứng với server OpenResty. Do đó, mọi request HTTP/HTTPS trên cổng 80/443 được chuyển đến từ các tên miền `dvwa.test` và `nginx.test` sẽ chuyển tiếp đến server OpenResty. Bước tiếp theo, chúng ta sẽ cấu hình việc chuyển tiếp HTTP request từ server OpenResty đến 2 server DVWA và Nginx.

3.3.4 Cấu hình reverse proxy cho OpenResty server

Trong bước này, sau khi các HTTP request đến DVWA và Nginx server đã được điều hướng hoàn toàn sang OpenResty server, để chuyển tiếp HTTP request đến server đích tương ứng, chúng ta sẽ chỉnh lại file cấu hình OpenResty tại đường dẫn `/usr/local/openresty/nginx/conf/nginx.conf` như bên dưới.

```

...
http {
    server {
        listen 80;
        listen 443 ssl;

        ...

        location / {
            resolver 127.0.0.11;

            set $target '';

            rewrite_by_lua_block {
                local host = ngx.var.http_host
                if host == "localhost" or
                    host:sub(-#"ntsec.cf") == "ntsec.cf"
                then
                    ngx.header.content_type = 'text/plain';
                    return ngx.say("Hello world!")
                end
                if host == "dvwa.test" then
                    ngx.var.target = "dvwa-test"
                elseif host == "nginx.test" then
                    ngx.var.target = "nginx-test"
                else
                    ngx.var.target = host
                end
            }

            proxy_pass http://$target;
        }

        ...
    }
}

```

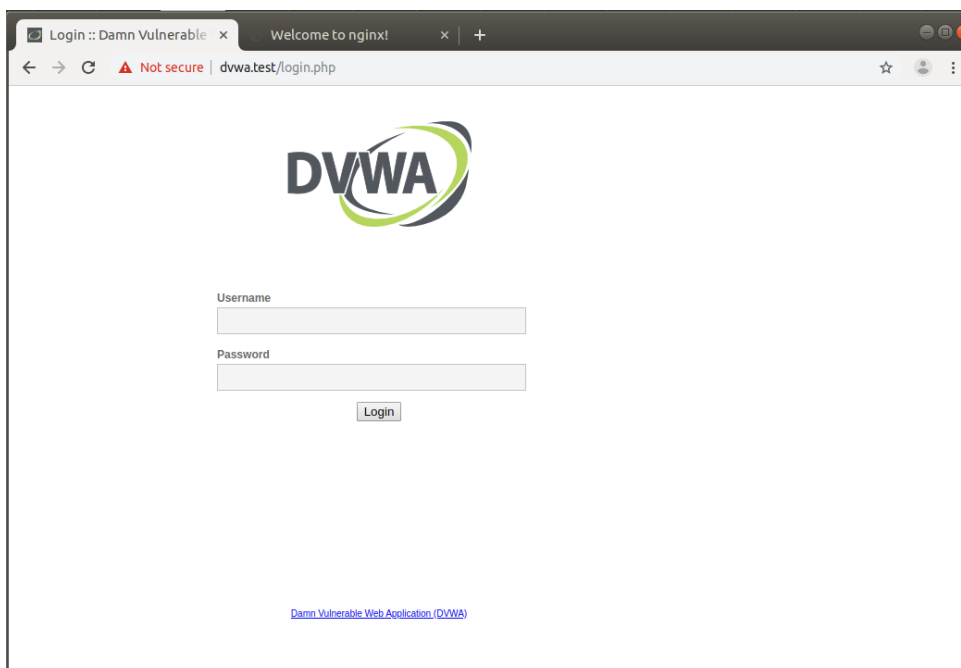
Giải thích: đoạn mã Lua đặt trong `rewrite_by_lua_block` làm nhiệm vụ chuyển hướng HTTP request dựa trên Host header trong HTTP request. Hình 3.6 là một HTTP request minh họa được gửi đến DVWA server. Dựa theo giá trị của Host header, request sẽ được chuyển hướng đến các Docker container tương ứng dựa theo tên trong file cấu hình `docker-compose.yml`. Ví dụ, với Host header là `nginx.test`, request sẽ được chuyển đến docker container `nginx-test`. Việc phân giải từ tên của Docker container sang địa chỉ IP sẽ được thực hiện thông qua Docker DNS server tại IP `127.0.0.11` thông qua chỉ thị cấu hình `resolver` trong file cấu hình OpenResty. Đối với các tên miền khác, Docker DNS server sẽ phân giải ra địa chỉ IP dựa theo cấu hình hệ thống phân giải DNS của máy chủ bên ngoài.

Sau khi chỉnh sửa file cấu hình OpenResty, ta cần khởi động lại OpenResty server và

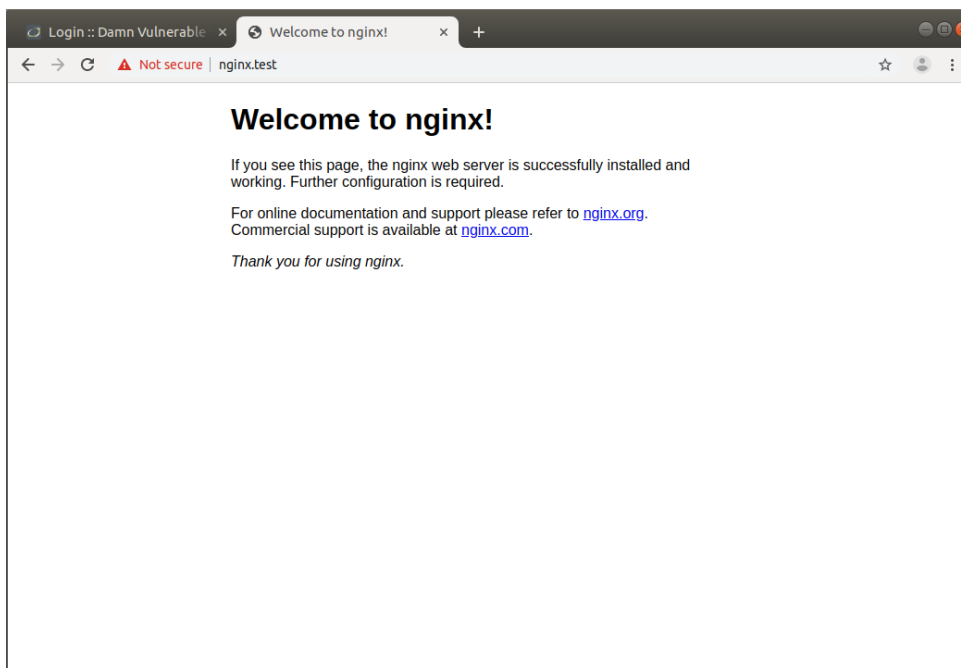
```
GET / HTTP/1.1
Host: dvwa.test
DNT: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
    like Gecko) Chrome/78.0.3904.108 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
    ,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Purpose: prefetch
Accept-Encoding: gzip, deflate
Accept-Language: vi-VN,vi;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: PHPSESSID=9nk22msdp09l68e337sh52dm16; security=low
Connection: close
```

Hình 3.6: HTTP Request tới http://dvwa.test

kiểm tra lại toàn bộ quá trình bằng cách vào trình duyệt và truy cập lần lượt tới các đường dẫn `http://dvwa.test`, `http://nginx.test`. Hình 3.7 và 3.8 là kết quả sau khi cài đặt thành công DVWA, Nginx test server và OpenResty với vai trò reverse proxy.



Hình 3.7: Kết quả sau khi truy cập đường dẫn `http://dvwa.test`



Hình 3.8: Kết quả sau khi truy cập đường dẫn `http://nginx.test`

3.3.5 Tích hợp ModSecurity CRS vào hệ thống

Các bước cấu hình cho server OpenResty về cơ bản đã hoàn thành, tiếp theo chúng ta sẽ tiến hành cài đặt ModSecurity CRS và bật ModSecurity WAF. Trước hết chúng ta tải mã nguồn của ModSecurity CRS từ trang github tại <https://github.com/SpiderLabs/owasp-modsecurity-crs> và đặt vào trong docker container chứa server OpenResty tại đường dẫn `/opt/owasp-modsecurity-crs` thông qua cấu hình `mount` trong file cấu hình của Docker compose. Ta đổi tên file cấu hình có sẵn của ModSecurity CRS từ `crs-setup.conf.example` sang `crs-setup.conf`, chúng ta sẽ chỉnh lại một số cấu hình dựa trên file có sẵn.

Cấu hình đầu tiên cần chỉnh lại là HTTP method, mặc định ModSecurity CRS chỉ cho phép người dùng gửi một số method thông dụng là `GET`, `HEAD`, `POST`, `OPTIONS`. Tuy nhiên, có một số ứng dụng tuân thủ theo chuẩn Restful API sẽ sử dụng cả một số method ít gặp như `PUT`, `PATCH`, `DELETE`. Do đó, cần bổ sung phần cấu hình sau để tránh bị *false positive*.

```
SecAction \
    "id:900200,\
    phase:1,\
    nolog,\
    pass,\
    t:none,\
    setvar:'tx.allowed_methods=GET HEAD POST OPTIONS PUT PATCH DELETE'"
```

Cấu hình quan trọng tiếp theo mà chúng ta cần chú ý là `paranoia_level`, cấu hình này dùng để điều chỉnh mức độ bảo mật của ModSecurity CRS như đã đề cập trong Chương 2. Chúng tôi sẽ giữ cấu hình này ở mức 1 như mặc định trong quá trình kiểm tra cài đặt ModSecurity CRS. Sau khi cài đặt thành công, chúng tôi sẽ chỉnh sang mức 4 nhằm thu thập tất cả các luật cho việc huấn luyện với học máy.

```
SecAction \
    "id:900000,\
    phase:1,\
    nolog,\
    pass,\
    t:none,\
    setvar:tx.paranoia_level=4"
```

Trong thư mục chứa ModSecurity CRS có một thư mục con cần chú ý là `rules`, đây là thư mục quan trọng nhất, chứa tất cả các tập luật mà ModSecurity CRS cung cấp. Ta có thể thêm, xóa hoặc chỉnh sửa các luật trong thư mục này cho phù hợp với ứng dụng cần bảo vệ.

Tiếp theo, chúng ta tạo file cấu hình cho ModSecurity WAF. File này sẽ được đặt tại đường dẫn `/etc/modsecurity/modsecurity.conf` (có thể thay đổi đường dẫn khác) trong server OpenResty. Chúng ta có thể tải file cấu hình được đề xuất bởi ModSecurity tại trang github ⁴. Trong cấu hình mặc định, khi phát hiện tấn công, ModSecurity chỉ ghi lại vào log file mà không chặn người dùng. Để dễ dàng kiểm tra xem ModSecurity đã được cài đặt thành công hay chưa, chúng ta sẽ tạm thời chỉnh sửa file cấu hình `/etc/modsecurity/modsecurity.conf` như bên dưới.

```
# SecRuleEngine DetectionOnly
SecRuleEngine On
```

Với cấu hình trên, ModSecurity sẽ chặn những request có điểm số bất thường cao, điểm bất thường được tính theo tổng điểm của tất cả các luật tương ứng. Cấu hình ngưỡng điểm bất thường có thể chỉnh sửa trong file `rules/REQUEST-901-INITIALIZATION.conf`.

```
# Default Inbound Anomaly Threshold Level (rule 900110 in setup.conf)
SecRule &TX:inbound_anomaly_score_threshold "@eq 0" \
    "id:901100,\
    phase:1,\
    pass,\
    nolog,\
    setvar:'tx.inbound_anomaly_score_threshold=5'"

# Default Outbound Anomaly Threshold Level (rule 900110 in setup.conf)
```

⁴<https://github.com/SpiderLabs/ModSecurity/blob/v3/master/modsecurity.conf-recommended>

```
SecRule &TX:outbound_anomaly_score_threshold "@eq 0" \
    "id:901110,\
    phase:1,\
    pass,\
    nolog,\
    setvar:'tx.outbound_anomaly_score_threshold=4'"
```

Bước cuối cùng, chúng ta bật ModSecurity bằng cách chỉnh sửa file cấu hình của OpenResty. Hai chỉ thị cần lưu ý ở đây là `modsecurity` và `modsecurity_rules_file`, đây là hai chỉ thị được thêm vào cấu hình của OpenResty bởi module connector ModSecurity-Nginx. Trong đó, chỉ thị `modsecurity` mang giá trị `on` hoặc `off`, dùng để bật và tắt ModSecurity WAF. Chỉ thị `modsecurity_rules_file` dùng để thiết lập đường dẫn của file cấu hình ModSecurity WAF.

```
http {
    server {
        listen 80;
        listen 443 ssl;
        modsecurity on;
        ...

        location / {
            resolver 127.0.0.11;

            set $target '';

            rewrite_by_lua_block {
                local host = ngx.var.http_host
                if host == "localhost" or
                    host:sub(-#"ntsec.cf") == "ntsec.cf"
                then
                    ngx.header.content_type = 'text/plain';
                    return ngx.say("Hello world!")
                end
                if host == "dvwa.test" then
                    ngx.var.target = "dvwa-test"
                elseif host == "nginx.test" then
                    ngx.var.target = "nginx-test"
                else
                    ngx.var.target = host
                end
            }

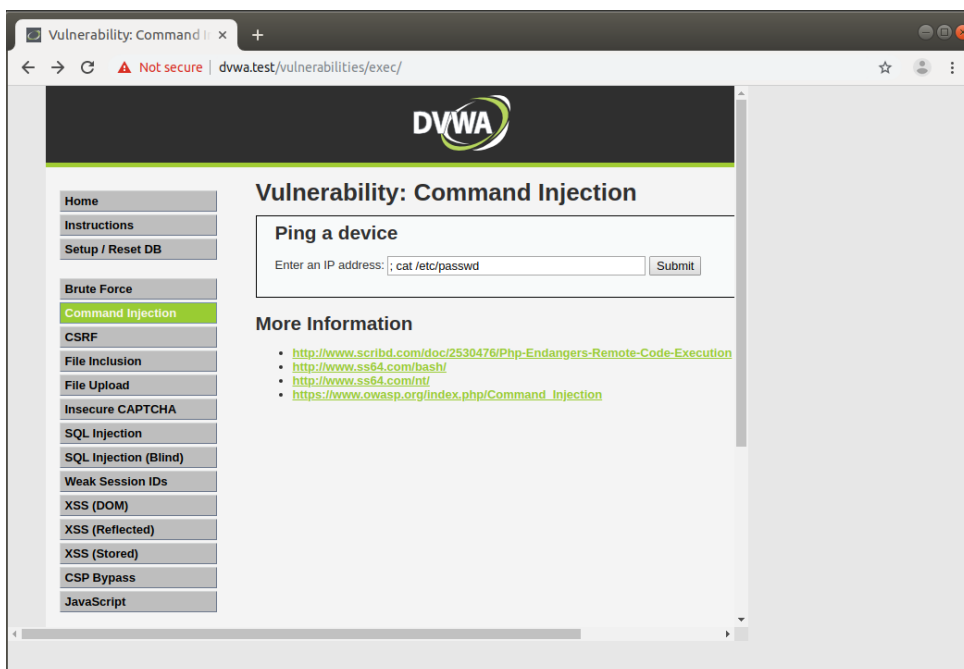
            proxy_pass http://$target;
            modsecurity_rules_file /etc/modsecurity/modsecurity.conf;
        }
        ...
    }
}
```

Để thêm các tập luật của ModSecurity CRS vào ModSecurity, ta cần thêm cấu hình sau vào file `/etc/modsecurity/modsecurity.conf`.

```
# setup crs
Include /opt/modsecurity-crs/crs-setup.conf

# import modsecurity crs rules dir
Include /opt/modsecurity-crs/rules/*.conf
```

Ta tiến hành khởi động lại OpenResty server và kiểm tra cài đặt của hệ thống WAF lần nữa bằng cách vào đường dẫn `http://dvwa.test` và thực hiện một tấn công bất kỳ như trong hình 3.9, kết quả nhận được sẽ là trang thông báo của OpenResty server với nội dung 403 Forbidden như trong hình 5.1.



Hình 3.9: Tấn công vào ứng dụng web DVWA

3.3.6 Chỉnh sửa thư viện ModSecurity

Hiện tại, ModSecurity chưa hỗ trợ việc lấy ra tất cả các luật ứng với từng request đầu vào. Do đó, để lấy ra được danh sách các luật trước khi ghi vào cơ sở dữ liệu log, chúng tôi phải tiến hành chỉnh sửa lại thư viện ModSecurity. Như đã đề cập ở Chương 2, ModSecurity có hỗ trợ một chỉ thị luật đặc biệt là `SecRuleScript`, chỉ thị này cho phép chúng ta sử dụng Lua script nhằm mục đích tạo ra các luật phức tạp hơn.

Để thuận tiện cho việc giải thích chỉ thị này, chúng tôi sẽ sử dụng một ví dụ tham khảo từ trang wiki của ModSecurity tại [https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-\(v2.x\)#SecRuleScript](https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-(v2.x)#SecRuleScript). Theo ví dụ tham khảo, cách sử dụng luật với chỉ thị này khá đơn giản như sau.

```
SecRuleScript "/path/to/file.lua" "block"
```

Và bên dưới là nội dung của Lua script.

```
function main()
    -- log a message with level 1 to audit log
    m.log(1, "Hello world!");

    -- access ModSecurity variable and apply transformation functions
    local var1 = m.getvar("REMOTE_ADDR");
    local var2 = m.getvar("ARGS", "lowercase");
    local var3 = m.getvar("ARGS.p", {
        "lowercase",
        "compressWhitespace"
    });
    -- do something here with var1, var2, var3

    return nil;
end
```

Dựa theo ví dụ trên, có một số điểm cần chú ý. Trước hết là hàm `main()`, đây là hàm sẽ được gọi bởi ModSecurity khi luật tương ứng được thực thi. Với giá trị trả về của hàm `main()` khác 0 hoặc `nil`, ModSecurity sẽ thực thi các hành động được định nghĩa trong luật bởi `SecRuleScript`, trong ví dụ này là “block”. Cũng như các luật khác, luật viết bằng chỉ thị `SecRuleScript` vẫn sẽ tuân thủ theo chu kỳ thực thi của ModSecurity (tham khảo lại Chương 2).

Điểm quan trọng tiếp theo, ModSecurity sẽ đăng ký một biến có tầm vực toàn cục là `m` trước khi thực thi hàm `main()` trong file Lua script. Các API mà ModSecurity cung cấp như `getvar()` hay `log()` trong ví dụ trên đều có thể truy cập thông qua biến toàn cục `m`. Việc đăng ký biến toàn cục `m` trong thư viện ModSecurity được thực hiện tại hàm `Lua::run()` trong file `src/engine/lua.cc`.

```
int Lua::run(Transaction *t, const std::string &str) {
#ifdef WITH_LUA
    std::string luaRet;
    const char *a = NULL;
    int ret = true;
    lua_State *L = luaL_newstate();
    luaL_openlibs(L);
```

```

luaL_newmetatable(L, "luaL_msc");
lua_newtable(L);

lua_pushlightuserdata(L, reinterpret_cast<void *>(t));
lua_setglobal(L, "__transaction");

luaL_setfuncs(L, mscLuaLib, 0);
lua_setglobal(L, "m");
...

```

Dựa theo mã nguồn của ModSecurity, hàm `Lua::run()` sẽ được gọi cho mỗi lần thực thi Lua script, tức là khi gặp chỉ thị `SecRuleScript` hoặc hành động `exec`. Ứng với mỗi luật `SecRuleScript`, một Lua engine sẽ được tạo, và môi trường của từng Lua engine sẽ phải khởi tạo lại trong mỗi lần thực thi. Do đó, nếu trong Lua script thực hiện nhiều tác vụ như import và gọi thư viện ngoài, việc này sẽ lặp đi lặp lại cho mỗi request và làm giảm hiệu năng do chưa có cơ chế caching ở mức module. Đây là một điểm yếu của việc dùng Lua script trong ModSecurity, do thời gian có hạn nên chúng tôi sẽ không tập trung vào việc thêm tính năng cache module cho thư viện ModSecurity.

Quay trở lại với vấn đề chính, nhằm hỗ trợ việc lấy ra các luật tương ứng với request đầu vào, chúng tôi sẽ hiện thực và bổ sung tính năng này vào ModSecurity Lua API. Cụ thể hơn, chúng tôi sẽ hiện thực hàm `getTriggeredRules()` và thêm vào biến `m`. Từ trong Lua script, khi cần lấy các luật trong ModSecurity chỉ cần gọi thông qua `m.getTriggeredRules()`. Chi tiết hiện thực hàm này khá đơn giản, có thể tham khảo mã nguồn của chúng tôi tại <https://github.com/ngoactint11vc/ModSecurity/blob/thesis/src/engine/lua.cc#L428>.

Danh sách toàn bộ các hàm được cung cấp bởi ModSecurity sau khi chỉnh sửa như sau. Có thể tham khảo mã nguồn tại <https://github.com/ngoactint11vc/ModSecurity/blob/thesis/src/engine/lua.h#L93>.

```

#ifdef WITH_LUA
static const struct luaL_Reg mscLuaLib[] = {
    { "log", Lua::log },
    { "getvar", Lua::getvar },
    { "getvars", Lua::getvars },
    { "setvar", Lua::setvar },
    { "getTriggeredRules", Lua::getTriggeredRules },
    { NULL, NULL }
};
#endif

```

Ngoài ra, có một vài trục trặc nhỏ trong thư viện ModSecurity trong quá trình chúng tôi thử nghiệm. Việc lấy ra các biến `REQUEST_HEADERS` và `RESPONSE_HEADERS` thông qua hàm `m.getvars()` có vẻ như gặp một số lỗi. Để sử dụng được 2 biến này nhằm lấy ra danh sách các header của HTTP request và response, chúng tôi đã có một sửa đổi nhỏ trong

file `src/variables/variable.h`, có thể tham khảo tại commit <https://github.com/ngoactint11vc/ModSecurity/commit/fad6c3a12f5967db83557843fb7a56d71ab3872e>. Có thể thấy, tính năng Lua script trong ModSecurity vẫn chưa được hoàn thiện, tài liệu hướng dẫn trên trang wiki của ModSecurity vẫn chưa đề cập cách sử dụng Lua API một cách chi tiết.

3.3.7 Cấu hình cơ sở dữ liệu lưu log

Tiếp theo, chúng tôi sẽ cài đặt hệ cơ sở dữ liệu MongoDB vào một Docker container dùng cho việc lưu HTTP request log mà server OpenResty WAF thu thập được. Công việc đầu tiên là dựng lên server MongoDB, chúng tôi sử dụng một Docker image có sẵn của MongoDB từ trang Docker Hub tại đường dẫn https://hub.docker.com/_/mongo. MongoDB là một hệ cơ sở dữ liệu theo hướng NoSQL, dữ liệu đầu vào không nhất thiết phải có cấu trúc sẵn, mỗi điểm dữ liệu có thể có cấu trúc khác nhau. Ngoài ra, do đặc tính không cấu trúc nên việc thao tác trên các dữ liệu lớn và có đặc tính không nhất quán sẽ trở nên dễ dàng hơn. Tốc độ thêm, xóa hoặc chỉnh sửa cũng nhanh hơn do không cần phải bảo đảm tính nhất quán dữ liệu như những hệ cơ sở dữ liệu SQL truyền thống, điển hình là MySQL. Dữ liệu HTTP request log sau khi đi vào server OpenResty WAF sẽ được ghi lại và chuyển đổi sang dạng JSON ⁵ trước khi ghi vào MongoDB Log. Ứng với mỗi HTTP request, chúng tôi sẽ chuyển dữ liệu theo cấu trúc mà chúng tôi định nghĩa như bên dưới.

```
{
  _id: ObjectId('5ecbcc6038494c73b31dd442'),
  request_body: '',
  request_method: 'GET',
  uri: '/tienda1/publico/pagar.jsp?modo=insertar&precio=1633&B1=
Confirmar\`INJECTED_PARAM`,
  response_status: 200,
  matched_rules: [
    {
      matched_value: 'Matched "Operator \'ValidateByteRange\' with
parameter \'38,44-46,48-58,61,65-90,95,97-122\' against variable \'ARGS
:B1\' (Value: \'Confirmar\`INJECTED_PARAM\` )',
      msg: 'Invalid character in request (outside of very strict
set)',
      id: 920273
    }
  ],
  remote_ip: '172.19.0.1',
  response_headers: {
```

⁵JSON (JavaScript Object Notation): là một chuẩn dữ liệu mở dùng trong lưu trữ và truyền tải, dữ liệu JSON là một dạng dữ liệu plain text với mục đích dễ đọc và dễ hiểu, mỗi JSON object bao gồm các cặp key-value và array. Là chuẩn dữ liệu rất phổ biến hiện nay, tham khảo thêm tại <https://en.wikipedia.org/wiki/JSON>

```

    ETag: '"5dd3e500-264"',
    Connection: 'close',
    'Content-Length': '612',
    'Accept-Ranges': 'bytes',
    'Content-Type': 'text/html',
    Date: 'Mon, 25 May 2020 13:47:12 GMT',
    'Last-Modified': 'Tue, 19 Nov 2019 12:50:08 GMT'
  },
  request_headers: {
    'User-Agent': 'Mozilla/5.0 (compatible; Konqueror/3.5; Linux)
KHTML/3.5.8 (like Gecko)',
    'Accept-Charset': 'utf-8, utf-8;q=0.5, *;q=0.5',
    'Accept-Encoding': 'x-gzip, x-deflate, gzip, deflate',
    Pragma: 'no-cache',
    'Cache-control': 'no-cache',
    Cookie: 'JSESSIONID=C5AD0960418F710338A99E5C525FFC48',
    Connection: 'close',
    Host: 'nginx.test',
    'Accept-Language': 'en',
    Accept: 'text/xml,application/xml,application/xhtml+xml,text/
html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5'
  }
}

```

Việc định nghĩa cấu trúc cho mỗi HTTP request chủ yếu vẫn dựa vào giá trị các header của giao thức HTTP. Ngoài ra, chúng tôi bổ sung thêm 1 trường khá quan trọng là `matched_rules`. Trường này sẽ lưu một mảng các luật của ModSecurity CRS ứng với mỗi request, request càng khớp với nhiều luật thì khả năng bị xác định là tấn công sẽ càng cao.

Ngoài ra, để giúp cho việc tìm kiếm, xóa, sửa dữ liệu trở nên trực quan, chúng tôi sẽ cài đặt ứng dụng web **Mongo Express**, đây là ứng dụng giúp quản lý hệ cơ sở dữ liệu MongoDB và được dùng tương đối phổ biến hiện nay. Với Mongo Express, chúng tôi sẽ dùng Docker image chính thức tải về từ https://hub.docker.com/_/mongo-express. Dưới đây là phần cấu hình của 2 service MongoDB và Mongo Express trong file `docker-compose.yml`.

```

version: "3.7"
services:
  ...
  log-db:
    image: "mongo"
    container_name: waf-log-db
    environment:
      MONGO_INITDB_ROOT_USERNAME: "${MONGO_INITDB_ROOT_USERNAME}"
      MONGO_INITDB_ROOT_PASSWORD: "${MONGO_INITDB_ROOT_PASSWORD}"
    volumes:
      - "log-db:/data/db"
    restart: always
  log-db-viewer:

```

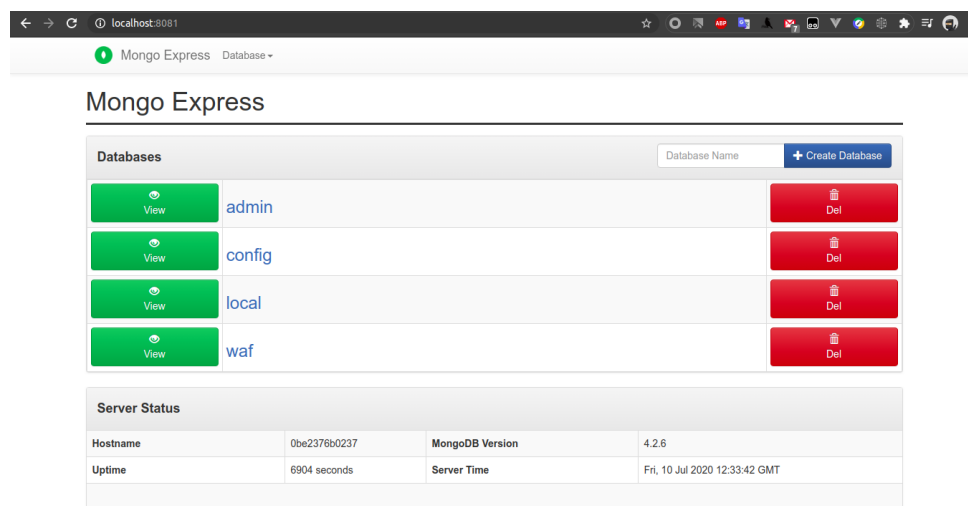


```

image: mongo-express
ports:
  - 127.0.0.1:8081:8081
environment:
  ME_CONFIG_MONGODB_SERVER: "${MONGODB_SERVER}"
  ME_CONFIG_MONGODB_ADMINUSERNAME: "${MONGO_INITDB_ROOT_USERNAME}"
  ME_CONFIG_MONGODB_ADMINPASSWORD: "${MONGO_INITDB_ROOT_PASSWORD}"
  ME_CONFIG_BASICAUTH_USERNAME: "${MONGO_EXPRESS_USERNAME}"
  ME_CONFIG_BASICAUTH_PASSWORD: "${MONGO_EXPRESS_PASSWORD}"
restart: always
volumes:
  log-db:
  ...

```

Để kiểm tra các service MongoDB và Mongo Express sau khi cài đặt, chúng ta vào đường dẫn <http://localhost:8081> trên trình duyệt. Nếu như cài đặt thành công, giao diện trên trình duyệt sẽ hiển thị như trong hình 3.10.



Hình 3.10: Giao diện ứng dụng Mongo Express sau khi cài đặt thành công

Sau khi đã cấu hình thành công server MongoDB dùng để làm cơ sở dữ liệu lưu log, việc tiếp theo cần làm là kết nối server OpenResty WAF đến MongoDB để gửi log. Trước hết, chúng tôi sẽ thêm một luật bổ sung vào thư mục chứa các luật của ModSecurity CRS tại đường dẫn `/opt/modsecurity-crs/custom-rules.conf`.

```

SecRuleScript "/opt/modsecurity-crs/lua-scripts/log_transaction.lua" \
  "id:999999002,\
  phase:5,\
  msg:'Transaction logged script run'"

```

Luật mà chúng tôi vừa bổ sung là một luật sử dụng chỉ thị `SecRuleScript`, trong đó sẽ thực thi file Lua script tại đường dẫn `/opt/modsecurity-crs/lua-scripts/log_transaction.lua`. Một điểm cần lưu ý là luật này sẽ được thực hiện trong giai đoạn 5 (có thể thao khảo lại Chương 2), đây là giai đoạn log trong ModSecurity, nếu thực hiện luật này trong các giai đoạn sớm hơn sẽ dẫn đến việc một số biến trong ModSecurity có giá trị không xác định. Nhiệm vụ của Lua script `log_transaction.lua` là lấy ra các luật trong ModSecurity CRS ứng với từng request, sau đó chuyển đổi request sang cấu trúc mà chúng tôi định nghĩa ở trên và ghi vào cơ sở dữ liệu log MongoDB. Nội dung của file Lua script có thể tóm tắt như bên dưới, tham khảo thêm tại https://github.com/ngoactint1lvc/waf/blob/master/openresty/modsecurity-crs/lua-scripts/log_transaction.lua.

```
...
local mongo = require("mongo")
local client = mongo.Client('mongodb://' .. MONGODB_USER .. ':' ..
    MONGODB_PASSWORD .. '@' .. MONGODB_SERVER .. '/admin')

local waf_normal_collection = client:getCollection('waf',
    collection_prefix .. 'normal_log')
...

function main()
    ...
    local request_headers = {}
    for i, header in ipairs(m.getvars("REQUEST_HEADERS")) do
        request_headers[header.name:sub("#REQUEST_HEADERS:" + 1)] =
header.value
    end

    local response_headers = {}
    for i, header in ipairs(m.getvars("RESPONSE_HEADERS")) do
        response_headers[header.name:sub("#RESPONSE_HEADERS:" + 1)] =
header.value
    end

    local transaction = {
        ["matched_rules"] = m.getTriggeredRules(),
        ["remote_ip"] = m.getvar("REMOTE_ADDR"),
        ["uri"] = m.getvar("REQUEST_URI"),
        ["request_method"] = m.getvar("REQUEST_METHOD"),
        ["request_headers"] = request_headers,
        ["request_body"] = m.getvar("REQUEST_BODY") or '',
        ["response_status"] = tonumber(m.getvar("RESPONSE_STATUS")),
        ["response_headers"] = response_headers
    }
    ...
    if waf_mode == 'LEARNING_NORMAL' then
        waf_normal_collection:insert(cjson.encode(transaction))
    end
end
```

```

    return 1
end

```

Các bước nêu trên đã trình bày tương đối chi tiết quá trình chuyển đổi dữ liệu từ dạng thô là HTTP request sang định dạng JSON kèm theo các luật của ModSecurity và quá trình lưu request log vào cơ sở dữ liệu.

3.3.8 Cấu hình forward proxy server

Bước cuối cùng, chúng ta sẽ tiến hành cấu hình một server làm nhiệm vụ forward proxy. Ở đây, chúng tôi sẽ tiếp tục sử dụng OpenResty web server. Đầu tiên, chúng ta dựng lên service proxy bằng cách thêm cấu hình sau trong file `docker-compose.yml`.

```

version: "3.7"
services:
  ...
  proxy-server:
    build:
      context: ./proxy-server
    volumes:
      - "./proxy-server/nginx.conf:/usr/local/openresty/nginx/conf/nginx.conf"
      - "./proxy-server/supervisor:/etc/supervisor"
      - "./proxy-server/run.sh:/run.sh"
    restart: always
    depends_on:
      - openresty
    ports:
      - "127.0.0.1:3004:80"
  ...
...

```

Mặc định, Nginx hay OpenResty đều chưa hỗ trợ method `CONNECT`, đây là method cho phép việc chuyển tiếp traffic TLS thông qua giao thức HTTP. Do đó, chúng tôi sẽ cài đặt thêm một module ngoài để hỗ trợ tính năng này, mã nguồn của module này có thể tham khảo tại https://github.com/chobits/nginx_http_proxy_connect_module.

Việc biên dịch và cài đặt module vào OpenResty được thực hiện thông qua các lệnh sau.

```

./configure -j4 \
  --with-pcre-jit \
  --with-compat \
  --with-http_ssl_module \
  --with-http_realip_module \
  --with-debug \
  --add-dynamic-module=/root/nginx_http_proxy_connect_module-0.0.1

```

```
make
make install
```

Sau khi biên dịch và cài đặt xong module `ngx_http_proxy_connect`, ta thêm file cấu hình sau vào OpenResty tại đường dẫn `/usr/local/openresty/nginx/conf/nginx.conf`.

```
load_module modules/ngx_http_proxy_connect_module.so;
...

http {
    ...
    server {
        listen 80;
        resolver 127.0.0.1 ipv6=off;
        server_name localhost;

        # forward proxy for CONNECT request
        proxy_connect;
        proxy_connect_allow      443;
        proxy_connect_connect_timeout 300s;
        proxy_connect_read_timeout 300s;
        proxy_connect_send_timeout 300s;

        # forward proxy for non-CONNECT request
        location / {
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;

            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_pass $scheme://openresty;
        }
    }
}
```

Cần lưu ý, cấu hình `proxy_pass` chỉ có tác dụng đối với các method không phải `CONNECT`, trong trường hợp này là traffic HTTP. Đối với traffic HTTPS, chúng ta không thể kiểm soát được server đích đến, do đó chúng ta cần điều hướng toàn bộ tên miền trong Docker container đến IP của server OpenResty WAF. Để chuyển hướng DNS như trên, chúng tôi cài đặt `Dnsmasq`, đây là một phần mềm hỗ trợ việc điều chỉnh DNS trở nên dễ dàng hơn. Sau khi cài đặt `Dnsmasq`, chúng ta thêm file cấu hình vào đường dẫn `/etc/dnsmasq.d/custom.conf` với nội dung.

```
address=#[Openresty_WAF_IP]
```

Ngoài ra, chúng ta cần chỉnh sửa lại file `/etc/resolv.conf` để điều chỉnh hệ thống

phân giải DNS mặc định của Docker container sang Dnsmasq. Nội dung file cấu hình `/etc/resolv.conf` như sau.

```
nameserver 127.0.0.1
```

Chúng ta khởi động container proxy server vừa mới cấu hình, sau đó đi đến đường dẫn `http://dvwa.test` trên trình duyệt thông qua địa chỉ HTTP proxy `http://localhost:3004` để kiểm tra. Nếu chúng ta vào được trang chính của ứng dụng DVWA tức là đã cài đặt và cấu hình thành công forward proxy server.

Cuối cùng, sau khá nhiều bước, chúng ta đã hoàn thành việc thiết kế và cài đặt hệ thống WAF cùng môi trường để phục vụ kiểm thử. Trong phần tiếp theo, chúng tôi sẽ bàn về một số hướng tiếp cận nhằm cải thiện hệ thống WAF hiện tại.

3.4 Một số hướng tiếp cận để cải thiện hệ thống WAF

Trong quá trình tìm hiểu, nghiên cứu, chúng tôi đã gặp khá nhiều phương pháp tiếp cận khác nhau. Mặc dù các phương pháp tiếp cận có thể đa dạng, kỹ thuật có thể khác nhau, nhưng về cơ bản có thể chia ra làm hai hướng tiếp cận chính là *blacklist* và *whitelist*.

Blacklist là phương pháp liệt kê toàn bộ các trường hợp không cho phép trong đầu vào, tức là phát hiện tấn công dựa vào một bộ cơ sở dữ liệu tấn công được định nghĩa sẵn hay còn gọi là signature-based. Phương pháp blacklist tương đối dễ dàng hiện thực và được áp dụng rộng rãi trong các hệ thống phát hiện xâm nhập (Intrusion detection system, IDS) cũng như các công cụ phát hiện virus phần mềm. Hướng tiếp cận dựa trên Blacklist được áp dụng khá rộng rãi trong các hệ thống WAF, điển hình là ModSecurity WAF và ModSecurity CRS, những tập luật dùng để phát hiện tấn công chính là một danh sách đen chứa các trường hợp bị cấm. Tuy nhiên, yếu điểm lớn nhất của phương pháp blacklist chính là không thể phát hiện được các loại tấn công mới (False Negative, FN) và trong nhiều trường hợp còn có thể hiểu lầm hành vi bình thường thành tấn công (False Positive, FP). Để hạn chế nhược điểm này, các bộ cơ sở dữ liệu tấn công phải được cập nhật thường xuyên và dữ liệu phải được chọn lọc cẩn thận bởi các chuyên gia bảo mật có kinh nghiệm.

Whitelist theo như tên gọi là một phương pháp ngược lại hoàn toàn với blacklist, thay vì dựa vào các dấu hiệu tấn công có sẵn, phương pháp này định nghĩa ra một tập các dữ liệu đầu vào được cho phép. Ưu điểm của phương pháp whitelist là giúp chúng ta tránh được “hầu hết” các loại tấn công có sẵn và cả những tấn công chưa từng biết (zero-day attack). Cơ sở của phương pháp này chính là ràng buộc những hành vi của người dùng theo những luật định nghĩa sẵn, điều này giúp cải thiện đáng kể khả năng chống chịu của hệ thống trước đa dạng các loại tấn công. Tuy nhiên, yếu điểm của phương pháp này cũng nằm ở việc ràng buộc hành vi của người dùng, điều này có thể gây ra những giới

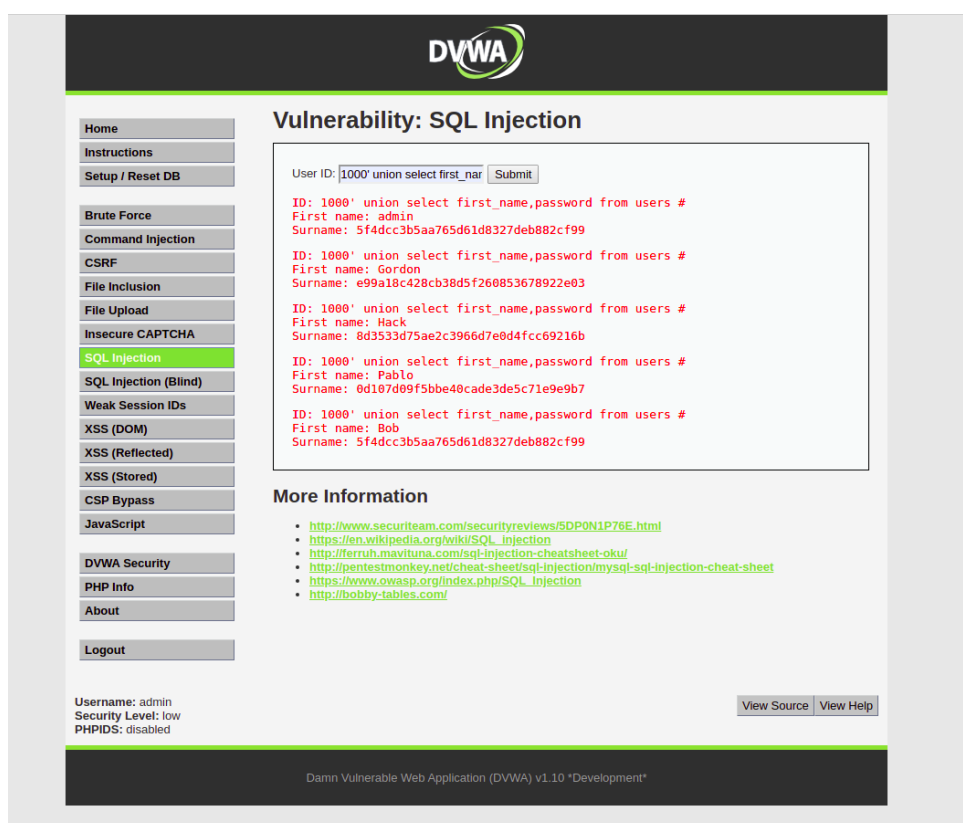
hạn về tính năng đối với người dùng. Ngoài ra, yếu điểm lớn nhất của phương pháp này là khó hiện thực, để xây dựng nên một tập các luật giúp tránh khỏi các loại tấn công khác nhau nhưng vẫn đảm bảo được các tính năng cần thiết cho người dùng là một công việc vô cùng phức tạp. Để dễ dàng diễn giải, chúng tôi sẽ lấy ví dụ đơn giản là hệ thống đăng nhập, trong đó có trường username và password. Chúng ta sẽ xem xét vector tấn công sau đây.

```
a" union select username, password from users #
```

Đối với những người có kiến thức cơ bản về bảo mật web sẽ dễ dàng nhận ra đây là một trường hợp đặc thù của tấn công SQL Injection. Trong trường hợp này, chúng tôi giả sử hệ thống đăng nhập đang bị lỗi SQL Injection, nếu nhập giá trị trên vào trường password thì sẽ gây ra lỗi về SQL trên hệ thống, dẫn tới nguy cơ bị xâm nhập và bị lộ toàn bộ cơ sở dữ liệu. Hình 3.11 là một ví dụ sau khi bị tấn công SQL injection thành công. Theo phương pháp whitelist, chúng ta phải giới hạn các ký tự cho phép được dùng trong trường password, ở đây chúng ta có thể giới hạn chỉ cho người dùng nhập ký tự chữ cái và số. Tuy nhiên điều này lại gây ra khá nhiều bất tiện cho những người dùng muốn dùng password chứa các ký tự đặc biệt nhằm chống lại tấn công vét cạn (brute-force). Đối với trường hợp này, phương pháp phát hiện tấn công dựa trên blacklist rõ ràng tỏ ra hiệu quả hơn so với whitelist.

Dựa vào hai ý tưởng chính được đề cập ở trên, nhiều phương pháp khác nhau đã được đề cập nhằm cải thiện các hệ thống WAF về cả mặt phòng thủ và tránh false positive. Đa số các phương pháp hiện tại được áp dụng vẫn hiện thực theo kiểu blacklist, trong đó ModSecurity CRS chính là một đại diện tiêu biểu. Thông thường WAF là một công cụ bảo vệ cho vô số ứng dụng web khác nhau từ ứng dụng blog, web bán hàng cho tới web ngân hàng, web doanh nghiệp. Sự đa dạng trong các thể loại ứng dụng web gây khó khăn đáng kể cho phương pháp tiếp cận dựa trên whitelist. Torrano et al. [15] đã đề cập về một phương pháp dựa trên hướng tiếp cận whitelist. Phương pháp này dùng một file cấu hình để định nghĩa các hành vi được phép trên ứng dụng web, tuy nhiên chỉ phù hợp cho từng ứng dụng web cụ thể bởi hành vi bình thường trên mỗi ứng dụng web sẽ khác nhau hoàn toàn. Có thể một request sẽ là bình thường với ứng dụng web này nhưng lại là bất bình thường với ứng dụng web khác. Về mặt kỹ thuật, đây sẽ là cách tốt nhất để bảo vệ ứng dụng web, tuy nhiên các hệ thống WAF hiện thực theo phương pháp này sẽ không thể mở rộng nhằm bảo vệ cho nhiều ứng dụng cùng lúc.

Trong những năm gần đây, các giải thuật học máy (machine learning) đã tỏ ra vô cùng hiệu quả với nhiều bài toán phức tạp như nhận diện hình ảnh, xử lý ngôn ngữ tự nhiên, trò chơi,... Một số hướng tiếp cận ứng dụng các giải thuật học máy trong hệ thống WAF cũng đã được đề cập trong nhiều nghiên cứu gần đây. Trong đó, M.Zhang et al [17] và W.Rong et al [10] đã đề xuất một số phương pháp giúp phát hiện tấn công trong HTTP request với mô hình mạng neuron CNN. A.Makiou et al [7] đề xuất phương pháp để phát hiện tấn công SQL injection dựa vào mô hình Naive Bayesian. T.Krueger et al [6] đề xuất



Hình 3.11: Ứng dụng web DVWA bị lộ cơ sở dữ liệu bởi tấn công SQL injection

hệ thống WAF TokDoc, trong đó sử dụng mô hình **n-gram** để xác định hành vi bất thường.

Phương pháp tiếp cận thông qua việc kết hợp giữa ModSecurity và các thuật toán học máy cũng không hẳn là một phương pháp mới. G.Betarte et al [1] đã đề xuất một hệ thống WAF, trong đó có sự kết hợp giữa ModSecurity CRS và mô hình **n-gram**, kết quả đầu ra sẽ được quyết định bằng cách tổng hợp giữa ModSecurity CRS và mô hình học máy. Trong chương tiếp theo, chúng tôi sẽ đề xuất một phương pháp kết hợp giữa ModSecurity và học máy.

Ứng dụng học máy vào hệ thống

4.1 Sự kết hợp giữa ModSecurity CRS và học máy

Như đã đề cập ở Chương 1, ModSecurity CRS là một tập luật có độ FP khá cao. Khi tăng mức độ bảo vệ (paranoid level) lên càng cao thì số luật kiểm tra sẽ nhiều hơn và dẫn đến độ FP cũng lên càng cao, trong một số trường hợp có thể lên gần 40% [5].

Nguyên nhân dẫn đến phát hiện nhầm các request bình thường thành tấn công chủ yếu đến từ một số luật khá nghiêm ngặt trong ModSecurity CRS. Để minh họa, chúng ta sẽ xem xét luật có id là 920272 trong ModSecurity CRS, nội dung của luật này như sau.

```
SecRule REQUEST_URI|REQUEST_HEADERS|ARGS|ARGS_NAMES|REQUEST_BODY "
  @validateByteRange 32-36,38-126" \
    "id:920272,\
    phase:2,\
    block,\
    t:none,t:urlDecodeUni,\
    msg:'Invalid character in request (outside of printable chars below
ascii 127)',\
    logdata:'%{MATCHED_VAR_NAME}=%{MATCHED_VAR}',\
    tag:'application-multi',\
    tag:'language-multi',\
    tag:'platform-multi',\
    tag:'attack-protocol',\
    tag:'OWASP_CRS',\
    tag:'OWASP_CRS/PROTOCOL_VIOLATION/EVASION',\
    tag:'paranoia-level/3',\
    ver:'OWASP_CRS/3.2.0',\
    severity:'CRITICAL',\
    setvar:'tx.anomaly_score_pl3=+{%tx.critical_anomaly_score}'"
```

Cụ thể, luật này sẽ kiểm tra giá trị của một số trường trong HTTP request như uri, request header, các tham số nhập vào,... Nếu bất kỳ giá trị được kiểm tra nào có chứa các

ký tự mà giá trị trong bảng mã ascii không thuộc khoảng từ 32 đến 36 hoặc 38 đến 126, luật này sẽ được kích hoạt, tức là khả năng bị nghi ngờ tấn công sẽ cao lên. Luật này ban đầu được dùng để chống một số tấn công khai thác lỗi RCE (Remote code execution ¹) trên các ứng dụng web thông qua việc giới hạn ký tự đặc biệt. Tuy nhiên, trong quá trình quan sát dựa trên traffic thực tế hàng ngày, chúng tôi nhận ra luật này bị kích hoạt với rất nhiều HTTP request bình thường, tức là độ FP của luật này rất cao.

Hầu hết các luật như trên trong ModSecurity CRS thường ở mức 3 trở lên. Với các luật ở mức 2, các tấn công thông thường có thể bị chặn khá hiệu quả bởi ModSecurity CRS. Tuy nhiên, chỉ với mức 2 thì sẽ có những kiểu tấn công khác nhau nhằm vượt qua sự phát hiện của ModSecurity CRS. Ngoài ra, kể cả khi đang cấu hình ModSecurity ở mức 2, vẫn có những trường hợp phát hiện sai do trong các request bình thường của một số trang web xuất hiện một chuỗi ký tự đặc biệt, mà chuỗi ký tự này hoàn toàn trùng khớp với một vector tấn công có sẵn trong tập luật của ModSecurity CRS. Ở đây, chúng tôi sẽ lấy ví dụ một request bình thường trong một ứng dụng web thực tế dựa trên nền tảng blog medium.com, đây là một nền tảng phổ biến, cho phép chia sẻ các bài viết với nhiều danh mục khác nhau.

```
GET /t%C3%ACm-cve-kh%C3%B4ng-kh%C3%B3-d-a3d7bd82c37a?source=false----
HTTP/1.1
Host: nginx.test
Connection: keep-alive
DNT: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
    like Gecko) Chrome/83.0.4103.116 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
    ,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,vi-VN;q=0.8,vi;q=0.7,es;q=0.6
```

Bên dưới là dữ liệu log của HTTP request do hệ thống WAF sinh ra, bao gồm giá trị các HTTP header và các luật kích hoạt tương ứng.

```
{
  ['request_body'] = '',
  ['request_method'] = 'GET',
  ['uri'] = '/t m-cve-kh ng -kh -d-a3d7bd82c37a?source=false----',
  ['response_status'] = 200,
```

¹Remote code execution: là tấn công khai thác ứng dụng phía server thông qua việc chèn các câu lệnh độc hại vào ứng dụng, thực hiện các câu lệnh trên ứng dụng theo ý muốn kẻ tấn công. Đây là một trong những tấn công nguy hiểm bậc nhất đối với các ứng dụng web.

```
[ 'matched_rules' ] = {
  [1] = {
    [ 'matched_value' ] = 'Matched "Operator \'ValidateByteRange\' with
parameter \'32-36,38-126\' against variable \'REQUEST_URI\' (Value: \'/t\
xc3\xacm-cve-kh\xc3\xb4ng-kh\xc3\xb3-d-a3d7bd82c37a?sourcex=false
----' )',
    [ 'msg' ] = 'Invalid character in request (outside of printable
chars below ascii 127)',
    [ 'id' ] = 920272
  },
  [2] = {
    [ 'matched_value' ] = 'detected SQLi using libinjection.',
    [ 'msg' ] = 'SQL Injection Attack Detected via libinjection',
    [ 'id' ] = 942100
  },
  [3] = {
    [ 'matched_value' ] = 'Matched "Operator \'Rx\' with parameter
\'(?:/\\*!?!|\\*/|\\[\\;|--|--[\\s\\r\\n\\v\\f]|--[^-]*?-[\\^&-]#.*?[\\s\\r\\n\\v\\f
]|;?\\x00)' against variable \'ARGS:sourcex\' (Value: \'false----' )',
    [ 'msg' ] = 'SQL Comment Sequence Detected',
    [ 'id' ] = 942440
  },
  [4] = {
    [ 'matched_value' ] = 'Matched "Operator \'Rx\' with parameter \'\\W{4}\'
against variable \'ARGS:sourcex\' (Value: \'false----' )',
    [ 'msg' ] = 'Meta-Character Anomaly Detection Alert - Repetitive Non
-Word Characters',
    [ 'id' ] = 942460
  },
  [5] = {
    [ 'matched_value' ] = 'Matched "Operator \'Rx\' with parameter
\'((?:[~!@#\\$%\\^&*\\(\\)\\-\\+=\\{\\}\\[\\]\\|\\:;\\\"' ' <>][~!@#\\$
%\\^&*\\(\\)\\-\\+=\\{\\}\\[\\]\\|\\:;\\\"' ' <>]*?){2})' against variable \'
ARGS:sourcex\' (Value: \'false----' )',
    [ 'msg' ] = 'Restricted SQL Character Anomaly Detection (args): # of
special characters exceeded (2)',
    [ 'id' ] = 942432
  }
},
...
```

Từ dữ liệu log như trên, có thể thấy rằng chỉ với một request thông thường trong một website phổ biến như medium.com đã khiến cho ModSecurity kích hoạt tới 5 luật. Nếu phân loại request này dựa theo công thức cộng điểm mặc định của ModSecurity CRS thì request này khả năng cao sẽ bị chặn. Kết quả này có thể tệ hơn nữa nếu chúng tôi không loại bỏ giá trị Cookie² của request này trước khi đi qua phân tích của hệ thống ModSecurity WAF.

²Cookie: là một header trong giao thức HTTP, thường được dùng để lưu các giá trị mang tính chất định danh cho mỗi client trong trình duyệt.

Với danh sách các luật kích hoạt như trên, việc đưa ra một công thức nào đó để quyết định xem một request là tấn công hay bình thường trở nên khá khó khăn. Theo cách hiện tại của ModSecurity CRS, công thức này được quyết định dựa vào tổng điểm của tất cả các luật được kích hoạt. Tuy nhiên, việc gán điểm cho từng luật cũng như quyết định ngưỡng điểm để đưa ra quyết định cuối cùng là hoàn toàn dựa trên kinh nghiệm của người viết luật. Có thể nói rằng công thức này vô cùng “*cảm tính*”, do đó việc chặn nhầm người dùng bình thường là hoàn toàn có thể xảy ra.

Một câu hỏi được đặt ra là “Liệu có cách nào biết được nên bật, tắt hoặc ưu tiên một luật nào đó trong ModSecurity CRS để cải thiện những yếu điểm trên hay không?” Để giải quyết câu hỏi trên, nhằm tìm ra một công thức phù hợp để quyết định kết quả dựa trên dữ liệu đầu vào là các luật của ModSecurity CRS, chúng tôi sẽ sử dụng một hướng tiếp cận dựa vào học máy. Trong các phần kế tiếp, chúng tôi sẽ trình bày chi tiết về cách chuẩn bị dữ liệu, chọn lọc đặc tính cũng như cách huấn luyện và tích hợp mô hình học máy vào hệ thống WAF.

4.2 Chuẩn bị dữ liệu

4.2.1 Vấn đề với bộ dữ liệu HTTP CSIC 2010

Theo như kế hoạch ban đầu trong đề cương luận văn, bộ dữ liệu (*dataset*) được sử dụng để huấn luyện cho mô hình học máy là HTTP CSIC 2010 [2]. Tuy nhiên trong quá trình thử nghiệm ban đầu, thông qua quá trình kiểm định bằng tay cũng như tham khảo nguồn gốc của bộ dữ liệu, chúng tôi đã nhận ra một số điểm hạn chế của bộ dữ liệu này.

- Bộ dữ liệu này được sinh ra hoàn toàn bằng công cụ tự động, thông tin dữ liệu nhập được sinh ra có tính chất ngẫu nhiên. Do đó, dữ liệu có phần nào không mang tính thực tế.
- Toàn bộ HTTP traffic sinh ra chỉ thuộc về duy nhất 1 trang web bán hàng [4]. Do đó, bộ dữ liệu này có phần không mang tính tổng quát, điều này dẫn đến việc các mô hình học máy khi được huấn luyện với bộ dữ liệu này sẽ “hoàn toàn không phù hợp” khi áp dụng với các trường hợp thực tế bởi hành vi của các website trên Internet là vô cùng đa dạng. Có thể lấy ví dụ đơn giản, một website bán hàng sẽ có hành vi hoàn toàn khác với một website chuyên về mạng xã hội, nếu dữ liệu huấn luyện không đủ tổng quát sẽ dẫn đến hiểu nhầm hành vi người dùng bình thường thành tấn công.

Cũng cần nói thêm, đầu vào dữ liệu trong đề tài là HTTP traffic từ người dùng, về cơ bản đặc tính dữ liệu dạng này vô cùng “nhạy cảm” vì có thể chứa *email* hoặc *password* quan trọng của người dùng. Do đó, việc có một bộ dữ liệu thực tế được công bố miễn phí trên internet là cực kỳ khó khăn, nếu có thì khả năng cao là dữ liệu công bố bất hợp pháp. Hơn thế nữa, dữ liệu thu thập được từ các trang web vận hành thực tế sẽ hoàn

toàn không có nhãn. Việc gán nhãn cho các HTTP request là hợp lệ hay tấn công sẽ tốn khá nhiều thời gian, việc tìm người đánh nhãn cho dữ liệu dạng này là vô cùng khó khăn vì đòi hỏi người đánh nhãn phải có kiến thức nhất định về bảo mật ứng dụng web. Nhìn chung, dữ liệu dạng này sẽ khó xử lý rất hơn nhiều so với dữ liệu như hình ảnh, giọng nói, văn bản thông thường, dữ liệu mà đa số mọi người đều có khả năng đánh nhãn.

Nói tóm lại, dù có bất kỳ mô hình học máy nào mà kết quả vô cùng tốt đối với bộ dữ liệu trên, thì việc áp dụng mô hình này cho các trường hợp thực tế cũng hoàn toàn không thể. Với tham vọng tạo được một hệ thống WAF có thể dùng được cho các ứng dụng web thực tế, việc sinh ra một bộ dữ liệu phù hợp hơn là vô cùng quan trọng. Trong các phần kế tiếp, chúng tôi sẽ trình bày chi tiết về cách mà hiện tại chúng tôi áp dụng để sinh dữ liệu cho mô hình học máy.

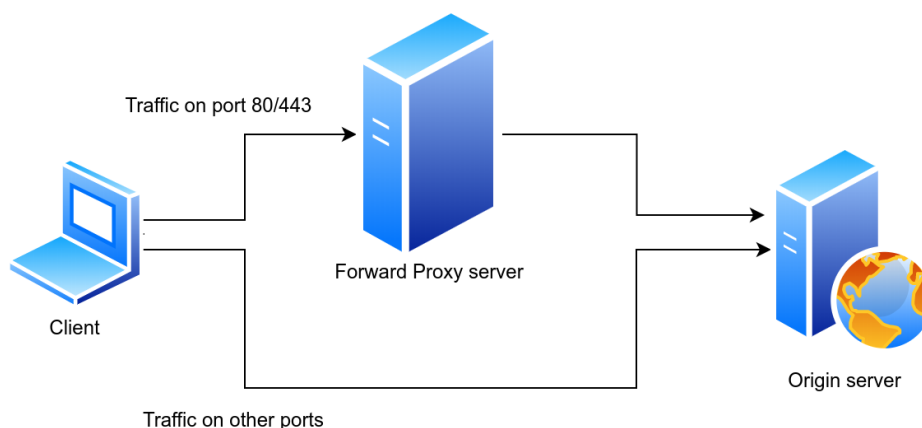
4.2.2 Sinh dữ liệu bình thường

Với mục đích tạo ra một bộ dữ liệu mà trong đó chỉ bao gồm các HTTP traffic bình thường, ý tưởng cơ bản ban đầu của chúng tôi là tự ghi lại toàn bộ traffic lướt web hàng ngày của mình. Tuy nhiên phải đảm bảo được trong suốt quá trình ghi lại traffic lướt web, toàn bộ request phải là bình thường. Cụ thể, tuyệt đối không sử dụng các công cụ dò quét hoặc khai thác lỗ hổng bảo mật web trong giai đoạn này, các công cụ này đa số sẽ sinh ra những traffic ở nhiều mức từ bất thường cho tới độc hại. Do đó sẽ ảnh hưởng đến dữ liệu để huấn luyện cho mô hình học máy.

Với ý tưởng trên, cách tiếp cận ban đầu của chúng tôi là can thiệp vào hệ thống DNS trong máy tính cá nhân của mình, chuyển hướng toàn bộ tên miền sang IP server WAF được cấu hình sẵn trong máy. Nói một cách đơn giản, chúng tôi sẽ cấu hình hệ thống DNS trong máy tính để khi truy cập vào một trang web bất kỳ như `google.com`, máy tính sẽ tự động phân giải ra IP của server WAF và chuyển hướng request đến. Ưu điểm của việc cấu hình chuyển hướng DNS như trên là vô cùng đơn giản, tuy nhiên có một nhược điểm lớn đã được chúng tôi phát hiện ra trong quá trình thử nghiệm ban đầu. Việc thay đổi DNS của toàn bộ máy sẽ dẫn đến việc các ứng dụng khác dựa trên các giao thức không phải HTTP sẽ không sử dụng được. Lấy ví dụ đơn giản, khi tên miền `example.com` bị trỏ về IP server WAF là `172.18.0.4`, lúc này nếu chúng ta sử dụng giao thức SSH để truy cập vào tên miền này thông qua cổng 22 sẽ gặp lỗi vì server WAF không hề hỗ trợ việc chuyển tiếp traffic TCP trên cổng 22.

Để khắc phục vấn đề này, chúng tôi đã tiến hành dựng thêm một server proxy. Trong đó, server proxy này sẽ hoạt động với vai trò là một forward proxy, lúc này toàn bộ HTTP traffic từ phía người dùng sẽ được chuyển đến server proxy này. Việc dựng và cấu hình forward proxy server này đã được chúng tôi đề cập ở Chương 3 trong mục “Cấu hình forward proxy server”. Tiếp theo đó, server proxy này sẽ chuyển tiếp toàn bộ HTTP request đến server WAF để kiểm tra và cuối cùng request sẽ được chuyển tiếp đến server đích mà người dùng yêu cầu. Có thể tham khảo lại phần này ở Chương 3 tại mục “Thiết

kế hệ thống”. Ưu điểm của việc cấu hình như trên là chúng ta có thể chỉ định traffic trên cổng nào hoặc tên miền nào sẽ được chuyển tiếp. Cụ thể ở đây, chúng tôi chỉ chuyển tiếp những traffic trên cổng 80 hoặc 443 tương ứng với 2 giao thức là HTTP và HTTPS đến server WAF, tất cả traffic trên những cổng còn lại sẽ đi trực tiếp đến server được yêu cầu như trong hình 4.1.



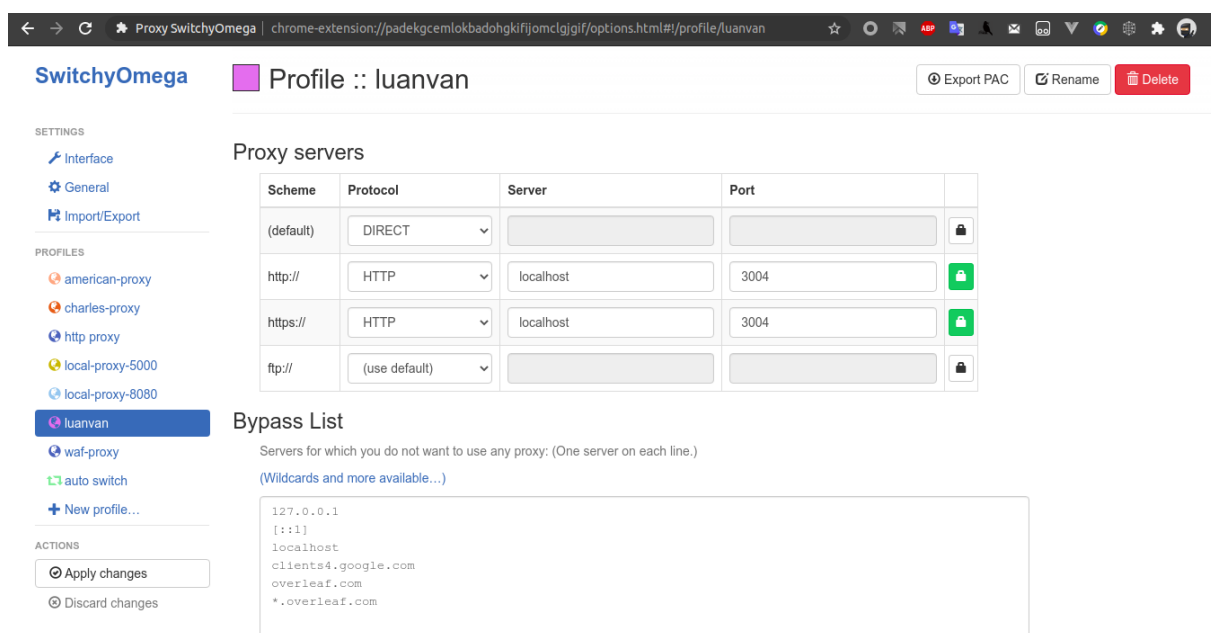
Hình 4.1: Chuyển tiếp traffic trên cổng 80/443 thông qua proxy server

Để làm được điều này, chúng tôi sử dụng một ứng dụng tiện ích mở rộng của trình duyệt Chrome là **Proxy SwitchyOmega**. Tiện ích này cho phép chúng tôi dễ dàng cấu hình proxy server cho từng tên miền thông qua kỹ thuật **Proxy Auto-Configuration**³. Trong hình 4.2 là giao diện cấu hình của tiện ích Proxy SwitchyOmega.

Một vấn đề tiếp theo cũng vô cùng quan trọng, đó là việc chuyển tiếp traffic HTTPS trên cổng 443. Như chúng ta đã biết, giao thức HTTPS chính là HTTP thông qua lớp mã hóa TLS. Để giải mã được traffic HTTPS trong quá trình request chuyển tiếp qua server WAF, chúng ta cần “giả mạo” chữ ký điện tử (certificate) của server đầu cuối mà người dùng đang truy cập. Việc đầu tiên cần làm là sinh ra một chứng chỉ CA (*Certificate Authority*)⁴, sau đó chúng tôi sẽ dùng chứng chỉ CA này để ký và phát hành các chữ ký điện tử cho từng website mà người dùng truy cập, những chữ ký này được gọi là **self-signed certificate**. Mặc định, các trình duyệt sẽ không cho phép người dùng kết nối đến các website sử dụng những chữ ký điện tử không được xác thực từ danh sách các CA hợp lệ nhằm mục đích chống tấn công **Man in the Middle (MITM)**. Tuy nhiên, hầu hết các trình duyệt vẫn cho phép người dùng tự thêm các chứng chỉ CA dành cho

³Proxy Auto-Configuration: tập tin cấu hình proxy tự động sử dụng ngôn ngữ JavaScript, tham khảo thêm tại [https://developer.mozilla.org/en-US/docs/Web/HTTP/Proxy_servers_and_tunneling/Proxy_Auto-Configuration_\(PAC\)_file](https://developer.mozilla.org/en-US/docs/Web/HTTP/Proxy_servers_and_tunneling/Proxy_Auto-Configuration_(PAC)_file)

⁴Certificate Authority: là cơ quan phát hành chữ ký điện tử, có một số lượng giới hạn chữ ký điện tử của CA được tin tưởng và cài đặt sẵn trong các trình duyệt, cho phép việc kiểm tra các chữ ký điện tử của các tổ chức khác



Hình 4.2: Giao diện của tiện ích Proxy SwitchyOmega

mục đích kiểm thử ở môi trường cục bộ, tính năng này chủ yếu được dùng bởi các nhà phát triển (developer). Do đó, chỉ cần thêm chứng chỉ CA tự tạo vào danh sách các CA tin tưởng (*Trusted CA*) trong trình duyệt thì có thể giải quyết được vấn đề trên.

Tuy nhiên, trong quá trình áp dụng, chúng tôi lại tìm ra được một phương pháp đơn giản hơn mà không cần phải tự tạo chữ ký điện tử cho mỗi tên miền. Trình duyệt Google Chrome mặc định đã có hỗ trợ tắt tính năng xác thực chữ ký điện tử của các website truy cập bằng cách bật cờ (flag) `--ignore-certificate-errors` khi khởi động. Với cách này, chỉ cần chỉnh lại câu lệnh khởi động trình duyệt Google Chrome như sau (dưới đây là lệnh khởi động trình duyệt Google Chrome trên hệ điều hành Linux).

```
/usr/bin/google-chrome-stable --ignore-certificate-errors
```

Cần lưu ý, việc tắt đi tính năng xác thực chữ ký điện tử cho các website có thể dẫn đến việc bị tấn công MITM, do đó sau khi thu thập dữ liệu xong cần bật lại tính năng này ngay lập tức.

Sau quá trình cấu hình như trên, việc còn lại là sử dụng trình duyệt Chrome để lướt web một cách bình thường. Tùy vào lưu lượng truy cập web hàng ngày mà số lượng request ghi lại có thể nhiều hoặc ít. Hiện tại, chúng tôi đã tự ghi lại traffic của mình trong vòng 5 ngày và đã thu được tổng cộng 182,103 HTTP request bình thường trên tổng số 1405 website khác nhau (tính theo Host header trong HTTP request). Kết quả thống kê 15 tên miền có lượt truy cập nhiều nhất trong tập dữ liệu của chúng tôi được thể hiện

trong bảng 4.1.

Bảng 4.1: Danh sách 15 tên miền có số lượng request nhiều nhất

Tên miền	Số lượng HTTP request
mail.google.com	28644
www.youtube.com	22776
www.google.com	15535
static.xx.fbcdn.net	10072
www.messenger.com	9115
play.google.com	4947
beacons.gcp.gvt2.com	4886
github.com	4224
github.githubassets.com	1894
i.ytimg.com	1818
calendar.google.com	1568
r3--sn-n5pbvoj5caxu8-nbos.googlevideo.com	1565
20.client-channel.google.com	1468
www.google-analytics.com	1453
edge-chat.messenger.com	1360

Do tính chất riêng tư và nhạy cảm của dữ liệu (có thể chứa access token, cookie, thông tin riêng tư và mật khẩu đăng nhập), chúng tôi sẽ không công bố chi tiết tập dữ liệu mà chúng tôi đã thu thập ở đây. Trong phần tiếp theo, chúng tôi sẽ trình bày cách mà chúng tôi sử dụng để sinh ra dữ liệu tấn công.

4.2.3 Sinh dữ liệu tấn công

Hiện tại, cách đơn giản nhất để sinh dữ liệu tấn công là sử dụng các công cụ mã nguồn mở có sẵn như SQLMap, NoSQLMap, MetaSploit,... Tuy nhiên, cần lưu ý về một số đặc điểm cũng như cơ chế hoạt động của các công cụ này. Hầu hết các công cụ đều tiến hành giai đoạn dò quét (scanning) nhằm tìm kiếm thêm thông tin từ trang web trước khi thực sự tiến hành tấn công. Do đó, trong số những request này vẫn có thể xen lẫn khá nhiều request bình thường, nếu không cẩn thận sẽ dẫn đến việc gán nhãn nhầm cho dữ liệu.

Chính vì vậy, muốn sử dụng các công cụ trên trước hết cần phải có cơ chế nào đó để loại bỏ đi các HTTP request bình thường. Sau quá trình cân nhắc và chọn lọc cẩn thận, cuối cùng, công cụ duy nhất mà chúng tôi chọn lựa để sinh dữ liệu tấn công là FTW. Công cụ này ban đầu được dùng như là một bộ kiểm thử đơn vị được dùng bởi ModSecurity CRS, trong bộ dữ liệu kiểm thử này hầu hết là các request chứa vector tấn công. Ngoài ra, do mang tính chất là một công cụ kiểm thử đơn vị nên có thể bảo đảm được dữ liệu sinh ra sẽ không bị xen lẫn giữa tấn công và bình thường. Bên dưới là định dạng của một testcase tấn công trong tập kiểm thử của ModSecurity CRS sử dụng công cụ FTW.

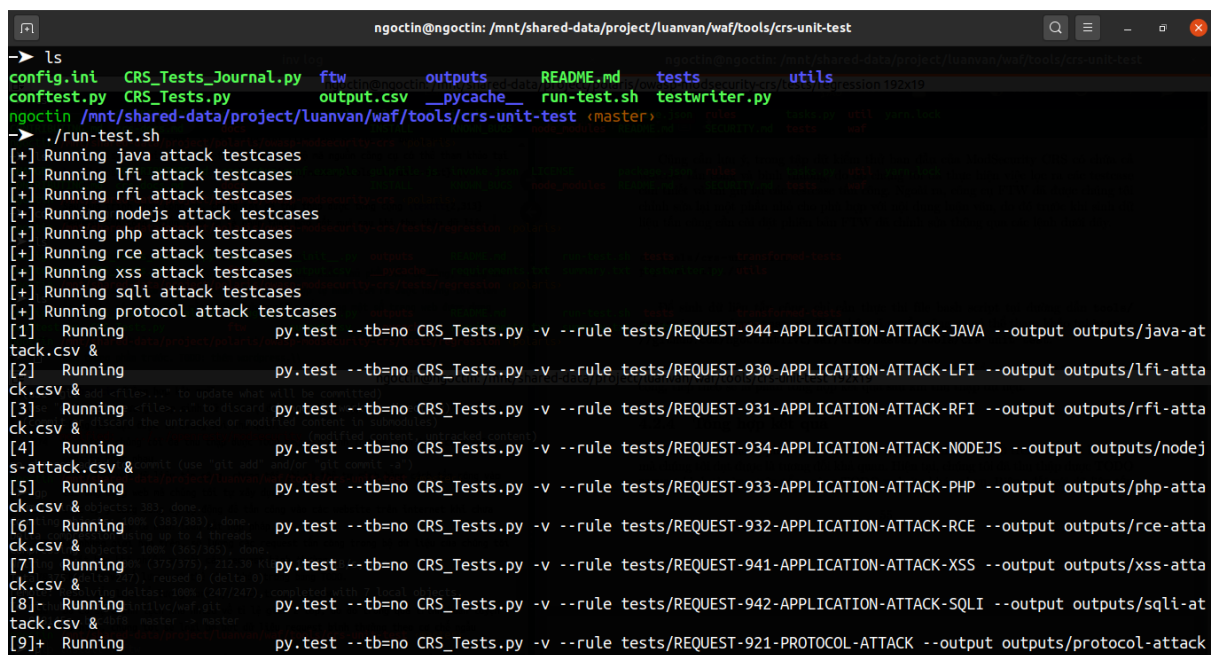
```
meta:
  author: csanders-git
  description: None
  enabled: true
  name: 933100.yaml
tests:
- desc: PHP Injection Attack (933100) from old modsec regressions
  stages:
  - stage:
    input:
      dest_addr: nginx.test
      headers:
        Accept: text/xml,application/xml,application/xhtml+xml,text/
html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
        Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
        Accept-Encoding: gzip,deflate
        Accept-Language: en-us,en;q=0.5
        Host: nginx.test
        Keep-Alive: '300'
        Proxy-Connection: keep-alive
        User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv
method: GET
port: 80
uri: /?foo=<?exec('wget%20http://r57.biz/r57.txt%20-0
version: HTTP/1.0
output:
  response_contains: 403 Forbidden
test_title: 933100-1
...
```

Cũng cần lưu ý, trong tập dữ liệu kiểm thử ban đầu của ModSecurity CRS có chứa cả testcase tấn công và bình thường, do đó chúng tôi đã thực hiện việc chọn lọc ra các testcase cần thiết và chỉ giữ lại các testcase tấn công. Ngoài ra, công cụ FTW đã được chúng tôi chỉnh sửa lại đôi chút cho phù hợp với mục tiêu của luận văn, do đó trước khi sinh dữ liệu tấn công cần cài đặt phiên bản FTW đã chỉnh sửa thông qua các lệnh dưới đây.

```
cd tools/crs-unit-test
pip install pytest
pip install ./ftw
```


Chú ý: để sử dụng công cụ trên, ta cần cài đặt Python và Pip (trình quản lý module của Python).

Để sinh dữ liệu tấn công, chỉ cần thực thi file bash script tại đường dẫn `tools/crs-unit-test/run-test.sh` như trong hình 4.3. Toàn bộ mã nguồn công cụ có thể tham khảo tại <https://github.com/ngoctint1lvc/waf/tree/master/tools/crs-unit-test>.



```
ngoctin@ngoctin: /mnt/shared-data/project/luanvan/waf/tools/crs-unit-test
-> ls
config.ini  CRS_Tests_Journal.py  ftw  outputs  README.md  tests  utils
confptest.py CRS_Tests.py          output.csv  __pycache__  run-test.sh  testwriter.py
ngoctin@mnt/shared-data/project/luanvan/waf/tools/crs-unit-test (master)
-> ./run-test.sh
[+] Running java attack testcases
[+] Running lfi attack testcases
[+] Running rfi attack testcases
[+] Running nodejs attack testcases
[+] Running php attack testcases
[+] Running rce attack testcases
[+] Running xss attack testcases
[+] Running sql attack testcases
[+] Running protocol attack testcases
[1] Running py.test --tb=no CRS_Tests.py -v --rule tests/REQUEST-944-APPLICATION-ATTACK-JAVA --output outputs/java-at
tack.csv &
[2] Running py.test --tb=no CRS_Tests.py -v --rule tests/REQUEST-930-APPLICATION-ATTACK-LFI --output outputs/lfi-atta
ck.csv &
[3] Running py.test --tb=no CRS_Tests.py -v --rule tests/REQUEST-931-APPLICATION-ATTACK-RFI --output outputs/rfi-atta
ck.csv &
[4] Running py.test --tb=no CRS_Tests.py -v --rule tests/REQUEST-934-APPLICATION-ATTACK-NODEJS --output outputs/nodej
s-attack.csv &
[5] Running py.test --tb=no CRS_Tests.py -v --rule tests/REQUEST-933-APPLICATION-ATTACK-PHP --output outputs/php-atta
ck.csv &
[6] Running py.test --tb=no CRS_Tests.py -v --rule tests/REQUEST-932-APPLICATION-ATTACK-RCE --output outputs/rce-atta
ck.csv &
[7] Running py.test --tb=no CRS_Tests.py -v --rule tests/REQUEST-941-APPLICATION-ATTACK-XSS --output outputs/xss-atta
ck.csv &
[8] Running py.test --tb=no CRS_Tests.py -v --rule tests/REQUEST-942-APPLICATION-ATTACK-SQLI --output outputs/sql-atta
ck.csv &
[9] Running py.test --tb=no CRS_Tests.py -v --rule tests/REQUEST-921-PROTOCOL-ATTACK --output outputs/protocol-attack
```

Hình 4.3: Công cụ sinh dữ liệu tấn công

Sau quá trình chạy công cụ, chúng tôi đã thu thập được tổng cộng 2,313 request tấn công, tiếp theo sẽ là phần tổng hợp kết quả sau khi thu thập dữ liệu.

4.2.4 Tổng hợp kết quả

Với những phương pháp sinh dữ liệu tự động như đã đề cập ở trên, kết quả ban đầu mà chúng tôi đạt được là tương đối khả quan. Hiện tại, chúng tôi đã thu thập được 182,103 request bình thường trên tổng số 1405 website khác nhau. Riêng đối với dữ liệu tấn công, chúng tôi chỉ có thể tự sinh bằng cách tấn công vào các ứng dụng web mà chúng tôi tự xây dựng. Việc tự ý dùng công cụ tự động để tấn công vào các website trên Internet khi chưa được cho phép là hoàn toàn bất hợp pháp. Do những giới hạn trên, số lượng các request tấn công trong bộ dữ liệu của chúng tôi sẽ ít hơn khá nhiều so với các request bình thường, chỉ bao gồm 2,313 request. Chi tiết về số lượng của từng loại có thể tham khảo trong bảng 4.2.

Bảng 4.2: Thống kê số lượng trong tập dữ liệu

Nguồn	Thể loại	Số lượng
Tự sinh	Tấn công	2,313
Tự sinh	Bình thường	182,103
CSIC 2010	Tấn công	21,451
CSIC 2010	Bình thường	66,958

Từ tập dữ liệu hiện tại mà chúng tôi có được, đối với dữ liệu của lớp “bình thường” chúng tôi sẽ sử dụng hoàn toàn dữ liệu tự sinh ra, tập dữ liệu bình thường của bộ CSIC 2010 sẽ không được chúng tôi sử dụng do không mang tính thực tế. Đối với dữ liệu tấn công, do sự khan hiếm về dữ liệu, chúng tôi sẽ kết hợp tập dữ liệu tấn công mà chúng tôi sinh được với tập dữ liệu tấn công của bộ CSIC 2010. Tóm lại, tập dữ liệu mà chúng tôi sẽ sử dụng cho mô hình có thể tóm tắt trong bảng 4.3.

Bảng 4.3: Tập dữ liệu được sử dụng trong mô hình

Nguồn	Thể loại	Số lượng	Tỉ lệ %
Tự sinh, CSIC 2010	Tấn công	23,764	11,54%
Tự sinh	Bình thường	182,103	88,46%

4.3 Huấn luyện mô hình

Bắt đầu giai đoạn huấn luyện mô hình, ý tưởng cơ bản ban đầu của chúng tôi là giữ lại toàn bộ tập luật của ModSecurity CRS, đồng thời phát hiện được những luật nào hay chặn nhầm người dùng bình thường để giảm sự ảnh hưởng của các luật này. Với ý tưởng trên, chúng tôi đề xuất một phương pháp kết hợp giữa ModSecurity CRS và học máy. Theo như nguyên lý hoạt động của ModSecurity đã đề cập ở các chương trước, khi mỗi request được phân tích bởi ModSecurity WAF, một tập các luật nào đó của ModSecurity CRS sẽ kích hoạt cho request đó. Từ đó, với tập các luật được kích hoạt bởi ModSecurity CRS cùng với một số đặc tính của giao thức HTTP (hiện tại chỉ sử dụng HTTP method), chúng tôi sẽ lấy làm các đặc trưng (features) trong việc huấn luyện mô hình. Có khá nhiều phương pháp học máy có thể áp dụng được trong trường hợp này, ví dụ như Linear Classification, Support Vector Machine (SVM), Decision Tree hoặc Random Forest,...

Việc trích xuất đặc trưng cũng sẽ được thực hiện hoàn toàn giống nhau cho cả 3 phương pháp trên. Trong tập luật của ModSecurity CRS hiện tại mà chúng tôi sử dụng có tổng cộng 156 luật, ứng với mỗi luật trên chúng tôi sẽ ánh xạ thành một đặc trưng trong tập dữ liệu. Nếu một request khiến cho luật nào đó kích hoạt, chúng tôi sẽ gán cho đặc trưng này giá trị là 1, những luật không kích hoạt sẽ có giá trị ngược lại là 0. Ngoài ra, như đã đề cập trước đó, chúng tôi sẽ sử dụng cả HTTP request method làm đặc trưng huấn luyện. Các giá trị hiện tại có thể có của HTTP request method trong bộ dữ liệu thu thập được bao gồm GET, POST, HEAD, OPTIONS, PUT. Để xử lý các đặc trưng này, chúng tôi sử dụng kỹ thuật **One-Hot Encoding**. Đây là một kỹ thuật đơn giản, giúp số hóa giá trị của các biến phân loại (*categorical variable*) trước khi huấn luyện. Quá trình chuyển đổi dữ liệu từ cấu trúc trong cơ sở dữ liệu log (tham khảo Chương 3 mục “Cấu hình cơ sở dữ liệu lưu log”) sang dạng bảng để huấn luyện với mô hình học máy khá đơn giản, cấu trúc sau khi chuyển đổi sẽ như trong hình 4.4.

944130	943110	942100	942101	...	request_method_GET	request_method_POST	...
0	0	1	0		1	0	
0	1	0	0		1	0	
0	0	1	0		0	1	

Hình 4.4: Cấu trúc của dữ liệu trước khi huấn luyện với học máy

Để giải thích đơn giản, chúng ta sẽ lấy ví dụ hàng đầu tiên trong bảng ở hình 4.4, request này sau khi đi qua ModSecurity sẽ kích hoạt một số luật, trong đó có luật 942100 và có request method là GET. Sau khi đã chuyển đổi toàn bộ tập dữ liệu sang dạng bảng, chúng tôi sẽ tách dữ liệu ra thành hai tập huấn luyện và kiểm thử với tỉ lệ 2:1. Tuy nhiên, do sự chênh lệch đáng kể về tỉ lệ giữa hai lớp cần phân loại, chúng tôi sẽ phân tách sao cho đảm bảo trong tập huấn luyện và kiểm thử có tỉ lệ hai lớp đồng đều nhau, cách chia này được gọi là **Stratified Split**.

Trước hết, chúng tôi sẽ thử huấn luyện với mô hình Decision Tree. Decision Tree là một mô hình học máy mà kết quả đầu ra được quyết định dựa trên cấu trúc dữ liệu cây, trong đó mỗi node sẽ ứng với một câu hỏi, dựa vào kết quả của đầu vào cho câu hỏi đó sẽ dẫn đến các nhánh con khác nhau, quá trình này tiếp tục đến khi gặp node lá để quyết định được dữ liệu đầu vào thuộc lớp nào. Dựa vào tập dữ liệu huấn luyện, chúng tôi sẽ sinh ra cây Decision Tree tương ứng. Theo đó, có một vài thông số mà chúng tôi cần cân nhắc khi huấn luyện với mô hình Decision Tree như sau.

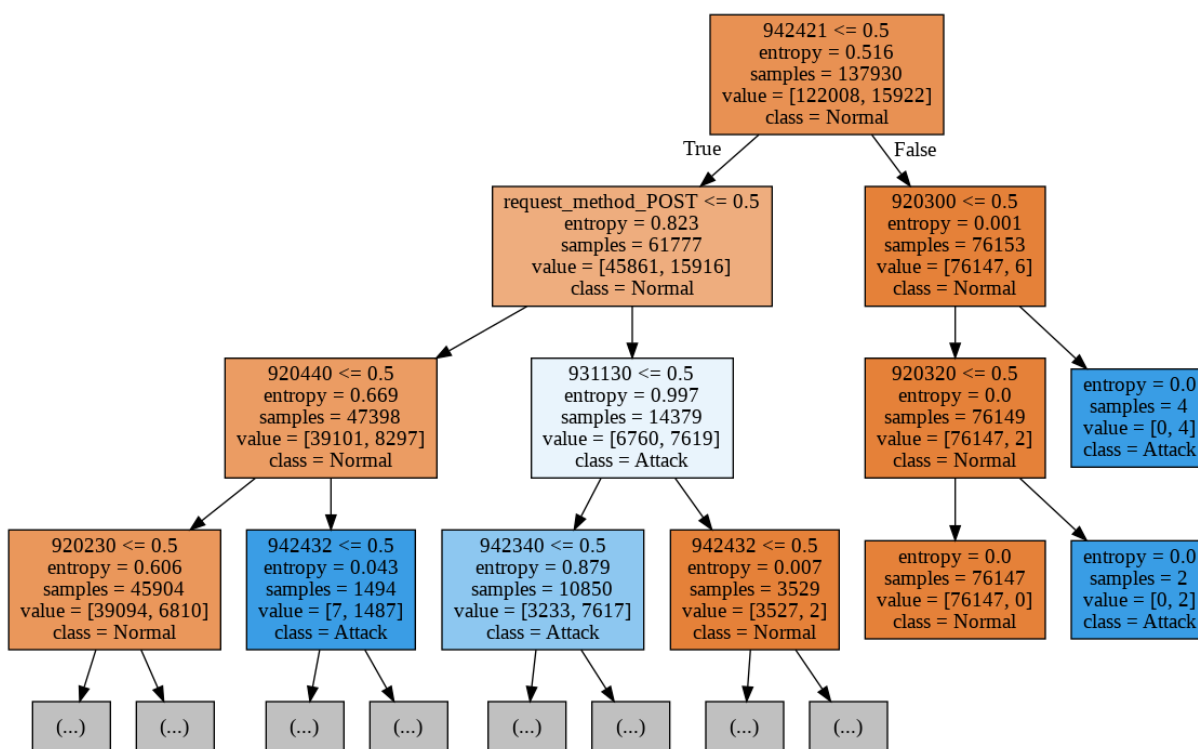
- **Độ đo khi phân tách nhánh.** Có 2 độ đo chủ yếu được dùng trong Decision Tree là **gini** và **entropy**, về cơ bản 2 độ đo này cho ra kết quả tương đối giống nhau. Độ đo **gini** thường được dùng thay thế cho **entropy** chủ yếu là để giảm đi các phép

tính toán phức tạp như logarit, với tập dữ liệu không quá lớn, chúng tôi sẽ chọn độ đo **entropy** trong quá trình huấn luyện mô hình nhằm tăng độ chính xác khi phân tách.

- **Độ sâu của cây.** Độ sâu của cây sẽ quyết định nhiều đến độ chính xác của mô hình, độ sâu càng lớn thì khả năng dự đoán của Decision Tree sẽ càng chính xác. Tuy nhiên, nếu độ sâu quá lớn cũng dễ dẫn đến việc **overfit**, tức là mô hình sẽ dự đoán vô cùng chính xác cho tập huấn luyện nhưng lại không mang tính tổng quát, không dự đoán chính xác cho các trường hợp chưa gặp. Ngoài ra, quan trọng hơn, cây có độ sâu lớn sẽ khiến mô hình phải tính toán nhiều, độ phức tạp của mô hình sẽ tăng ở mức lũy thừa trong khi độ chính xác tăng rất chậm. Trong thử nghiệm thực tế, với độ sâu kể từ 20 trở đi, chúng tôi quan sát thấy độ chính xác của mô hình không cải thiện nhiều. Do đó, chúng tôi sẽ giữ độ sâu của cây là 20.
- **Số mẫu tối thiểu ở node lá.** Ngoài cách giới hạn độ sâu của cây, chúng ta có thể giới hạn số mẫu tối thiểu ở một node lá của cây nhằm giảm độ phức tạp cho mô hình. Phương pháp này có ưu điểm là sẽ làm “mượt” cho mô hình, khiến cho mô hình không nhạy cảm với các phần tử ngoại lai (outlier). Tuy nhiên, do tỉ lệ giữa hai lớp trong dữ liệu đầu vào của chúng tôi không được cân bằng, sử dụng thông số này sẽ khiến cho kết quả bị lệch theo lớp có số lượng lớn hơn (cụ thể là lớp “bình thường”).
- **Tỉ lệ mẫu tối thiểu ở node lá.** Thông số này về cơ bản tương đối giống như “số mẫu tối thiểu ở node lá”, tuy nhiên thay vì giới hạn ở một con số cụ thể, thông số này giới hạn số mẫu ở node lá dựa theo một tỉ lệ cho trước theo số lượng của số mẫu trong tập huấn luyện. Ví dụ, có thể giới hạn số mẫu tối thiểu ở node lá bằng 0.0001% trên tổng số mẫu dữ liệu huấn luyện.
- **Cân bằng dữ liệu 2 mẫu.** Với số lượng mẫu bị lệch nhiều giữa hai lớp, một cách giải quyết có thể áp dụng là cân bằng số lượng hai lớp trước khi huấn luyện. Cụ thể, chúng ta có thể lặp lại những mẫu trong lớp có số lượng ít hơn sao cho số lượng mẫu của hai lớp bằng nhau. Tuy nhiên, đối với dữ liệu của chúng tôi, về bản chất thực tế, số lượng của hai lớp này hoàn toàn không cân bằng nhau. Cụ thể, trong dữ liệu log của một website thông thường, số lượng request bình thường sẽ nhiều hơn đáng kể so với số lượng request tấn công. Trong thử nghiệm thực tế, khi áp dụng phương pháp trên, tỉ lệ **false positive** hay nói cách khác là tỉ lệ hiểu nhầm request bình thường thành tấn công tăng lên khá đáng kể.

Với những thông số trên, sau quá trình huấn luyện chúng tôi thu được mô hình cây Decision Tree như trong hình 4.5. Dựa vào hình ảnh này, chúng ta có thể biết được request đầu vào được đánh nhãn nào dựa theo các luật ứng với request đó (có thể lấy từ cơ sở dữ liệu log). Kết quả của mỗi node trong cây có thể diễn giải khá đơn giản, ở đây chúng tôi sẽ lấy ví dụ với node gốc (root node). Dòng đầu tiên trên node gốc là biểu thức điều kiện “ $942421 \leq 0.5$ ”, điều kiện này ứng với câu hỏi “Liệu request có kích hoạt luật với id 942421 trong ModSecurity CRS hay không?”, dựa vào kết quả của request chúng ta sẽ

biết phải kiểm tra với node con nào tiếp theo. Tiếp theo đó là độ đo entropy của node, có thể hiểu đơn giản, độ đo entropy biểu diễn cho sự không chắc chắn trong việc dự đoán giá trị của nhãn dựa theo phân phối xác suất của từng nhãn. Độ đo entropy được giới hạn trong khoảng từ 0 đến 1, độ đo entropy càng cao thì chúng ta càng khó dự đoán kết quả đầu ra từ node đó, entropy bằng 0 thì kết quả dự đoán là hoàn toàn chắc chắn (chúng ta có thể xem ví dụ trong các node lá) và ngược lại. Các thông số còn lại đọc từ trên xuống tương ứng là số mẫu dữ liệu trong node (**samples**), số mẫu cụ thể của từng lớp (**value**) và lớp có số lượng nhiều nhất trong tập mẫu (**class**).



Hình 4.5: Mô hình cây Decision Tree sau khi huấn luyện

Một câu hỏi được đặt ra lúc này là “Tại sao không sử dụng những phương pháp phức tạp hơn như Neuron Network, thay vì những phương pháp học máy đơn giản như Decision Tree hay Random Forest?”. Câu trả lời rất đơn giản, việc ứng dụng Neuron Network là hoàn toàn có thể, một số hướng tiếp cận có thể kể đến như sau.

- Huấn luyện với Neuron Network với dữ liệu HTTP request dạng thô, cách này nếu áp dụng có thể giúp cho mô hình nhận diện được những tấn công chưa hề biết trước, thậm chí là nhận diện được những hành vi bất thường. Tuy nhiên, việc huấn luyện mô hình dựa trên dữ liệu thô sẽ dẫn đến một số kết quả không thể kiểm soát được. Có thể xảy ra trường hợp request bị chặn chỉ vì hơi khác biệt với dữ liệu trong tập bình thường trước đó. Ngoài ra, nếu dự đoán dựa trên dữ liệu HTTP request dạng thô sẽ làm ảnh hưởng khá nhiều đến hiệu năng của hệ thống. Việc áp dụng một

mô hình phức tạp có thể không phù hợp cho các hệ thống WAF phân tích HTTP request dạng realtime, mô hình như vậy chỉ phù hợp khi phân tích trên dữ liệu HTTP request log theo kiểu chạy ngầm (background).

- Sử dụng đầu vào là các luật của ModSecurity CRS như trên, chỉ thay phần mô hình Decision Tree bằng Neuron Network. Với khả năng xấp xỉ bất kỳ hàm số liên tục nào của mô hình Neuron Network (theo *Định lý xấp xỉ phổ dụng* hay *Universal approximation theorem*), đây là một cách tiếp cận tương đối hợp lý, có thể tìm ra được công thức quyết định cuối cùng cho ModSecurity CRS thay cho mô hình Decision Tree hiện tại. Tuy nhiên, do thời gian có hạn, và hướng này được chúng tôi tìm ra hơi muộn nên vẫn chưa kịp thử nghiệm.
- Ngoài ra, một cách tiếp cận nữa là chọn lọc ra những luật trong ModSecurity CRS và bổ sung thêm một tập luật có tính tổng quát hơn, những luật này cũng cần trả về giá trị cụ thể hơn thay vì chỉ là những giá trị **true** hoặc **false**. Ví dụ, một request kích hoạt một luật bao nhiêu lần hoặc có bao nhiêu keyword đặc biệt trong request. Ngoài ra, sau khi một chuỗi bị kích hoạt bởi luật nào đó, ta có thể đưa chuỗi đó vào một mô hình Neuron Network để quyết định xem luật có phát hiện nhằm chuỗi này thành tấn công hay không. Cách này có thể giới hạn lại miền tính toán của mô hình Neuron Network thay vì phải xử lý trên toàn bộ HTTP request. Với cách này, chúng ta cần phải kiểm tra và bổ sung khá nhiều luật cho ModSecurity CRS cũng như chỉnh sửa lại thư viện ModSecurity nhiều hơn. Tuy nhiên, đây là một hướng tiếp cận khá hứa hẹn.

Việc huấn luyện mô hình dựa trên kết quả của các tập luật cũng có một số điểm hạn chế, điển hình là không phát hiện được những tấn công mới không có trong tập luật. Tuy nhiên, việc sử dụng các tập luật cũng có nhiều ưu điểm như dễ dàng chỉnh sửa, kết quả đầu ra của mô hình hoàn toàn có thể hiểu được và tinh chỉnh một cách phù hợp. Ví dụ khi gặp một kết quả sai, đối với mô hình Decision Tree, chúng tôi có thể dễ dàng truy theo các nhánh từ gốc đến node lá và hoàn toàn hiểu được nguyên nhân dẫn đến kết quả sai, từ đó chúng tôi có thể chỉnh sửa lại mô hình một cách dễ dàng. Và trên hết, việc huấn luyện mô hình với Decision Tree hiện tại chỉ mất vài giây. Đối với mô hình Neuron Network huấn luyện trực tiếp từ dữ liệu thô mà không thông qua các tập luật, việc hiểu được mô hình là tương đối khó khăn, với một kết quả sai, chúng ta không có cách nào can thiệp vào mô hình để điều chỉnh lại. Do đó, chúng tôi vẫn đề cao các giải pháp dựa trên các tập luật hơn là Neuron Network bởi tính dễ hiểu, dễ điều chỉnh khi gặp các trường hợp ra kết quả không như mong đợi.

Ngoài ra, để cải thiện mô hình Decision Tree, chúng tôi đã tiến hành thử nghiệm với một mô hình kết hợp phổ biến là Random Forest. Về cơ bản, Random Forest là mô hình kết hợp quyết định dựa trên nhiều cây Decision Tree. Đối với Random Forest, có một vài thông số mà chúng tôi cần nhắc như sau.

- **Số lượng cây.** Về cơ bản, Random Forest sẽ quyết định một phần tử thuộc lớp nào dựa vào quyết định số đông của nhiều cây khác nhau, các cây này được xây dựng

thông qua việc loại bỏ và thay thế một vài điểm dữ liệu từ tập huấn luyện. Số cây càng lớn thì thời gian huấn luyện sẽ càng lâu, trong thử nghiệm thực tế, chúng tôi nhận thấy kể từ 50 cây trở lên, độ chính xác của mô hình không còn cải thiện nhiều.

- **Độ sâu của mỗi cây.** Random Forest về cơ bản là một mô hình được áp dụng để cải thiện độ *overfit* khi Decision Tree có độ sâu lớn, do đó chúng tôi sẽ không giới hạn độ sâu của mỗi cây khi huấn luyện với mô hình Random Forest.

Trong quá trình thử nghiệm, chúng tôi nhận thấy mô hình Random Forest không cải thiện kết quả được nhiều như chúng tôi mong đợi, phần kết quả về độ chính xác của hai mô hình này sẽ được chúng tôi đề cập cụ thể trong Chương 6. Toàn bộ quá trình huấn luyện mô hình học máy có thể tham khảo trong IPython Notebook của chúng tôi tại <https://github.com/ngoactint11vc/waf/blob/master/ml-model/train.ipynb>.

4.4 Tích hợp mô hình vào hệ thống

Sau quá trình huấn luyện, chúng tôi đã thu được hai mô hình là Decision Tree và Random Forest được lưu lại dưới dạng *serialized object* thông qua thư viện *Pickle*⁵. Quá trình *serialize* có thể hiểu là chuyển đổi một đối tượng (object) từ bộ nhớ thành một chuỗi byte để lưu vào file nhằm tái sử dụng lại sau này. Từ các mô hình đã lưu, chúng tôi sẽ chuyển đổi sang mã nguồn C thông qua thư viện *Scikit-Learn-Porter*⁶ nhằm tích hợp vào ModSecurity.

Mã nguồn của công cụ chuyển đổi mô hình có thể tham khảo trong thư mục `tools/ml_util`. Sau khi chuyển đổi mô hình sang mã nguồn C, chúng tôi sẽ hiện thực phần Lua binding như bên dưới và biên dịch ra file shared object với mục đích gọi trực tiếp từ file Lua script trong ModSecurity CRS.

```
static const struct luaL_reg funcs[] = {
    { "predict", l_predict },
    { NULL, NULL }
};

int luaopen_decision_tree(lua_State *L) {
    luaL_register(L, "decision_tree", funcs);
    return 0;
}
```

Dưới đây là một phần mã nguồn C được chuyển đổi từ mô hình Decision Tree.

⁵Pickle: thư viện dùng cho quá trình serialize Python object, tham khảo tại <https://docs.python.org/3/library/pickle.html>

⁶Scikit-Learn-Porter: mã nguồn của thư viện có thể tham khảo tại <https://github.com/nok/sklearn-porter>

```

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int predict(float features[159]) {

    int classes[2];

    if (features[128] <= 0.5) {
        if (features[157] <= 0.5) {
            if (features[138] <= 0.5) {
                if (features[45] <= 0.5) {
                    ...
                } else {
                    if (features[68] <= 0.5) {
                        ...
                    }
                }
            }
        }
    }

    int index = 0;
    for (int i = 0; i < 2; i++) {
        index = classes[i] > classes[index] ? i : index;
    }
    return index;
}

```

Dựa vào mã nguồn trên, có thể thấy `predict()` là hàm được sinh ra bởi thư viện `Sklearn Porter`, hàm này nhận input đầu vào là các luật của `ModSecurity CRS` như đã mô tả trong phần huấn luyện mô hình và đầu ra là kết quả tấn công hoặc bình thường. Để biên dịch ra các file shared object, chúng tôi sử dụng file `Makefile` như bên dưới.

```

CFLAGS= -Wall -shared -fPIC -I/usr/include/lua5.1 -llua5.1

all: decision_tree.so random_forest.so test

decision_tree.so: decision_tree.c
    gcc -o decision_tree.so $(CFLAGS) decision_tree.c

random_forest.so: random_forest.c
    gcc -o random_forest.so $(CFLAGS) random_forest.c

test:
    ls ./tests/*.lua | xargs -n 1 lua

clean:
    rm *.so

```

Kết quả sau khi biên dịch là hai file shared object có thể tìm thấy trong mã nguồn của chúng tôi tại thư mục `openresty/modsecurity-crs/lua-scripts/ml-model`. Tiếp theo, chúng tôi sẽ thêm một luật bổ sung vào file `openresty/modsecurity-crs/custom-rules`.

conf đã được thêm trước đó với nội dung như sau.

```
# After collected all trigger rules, using ML to evaluate final result
SecRuleScript "/opt/modsecurity-crs/lua-scripts/ml_evaluate.lua" \
    "id:999999001,\
    phase:2,\
    deny,\
    status:403,\
    msg:'Lua script triggered',\
    log,\
    auditlog"
```

Luật này sẽ thực thi file Lua script `ml_evaluate.lua`, nếu kết quả trả về từ hàm `main()` khác 0 hoặc nil, tức là mô hình học máy phát hiện request có khả năng là tấn công, ModSecurity sẽ chặn người dùng ngay lập tức với HTTP status code trả về 403. Nội dung của Lua script có thể tóm tắt như bên dưới.

```
local util = require("util")
local ml_model = require("random_forest")

function should_block(m, rules)
    local features = {
        '944130',
        '943110',
        ...
        'request_method_GET',
        ...
    }

    local request_method = m.getvar("REQUEST_METHOD")

    local features_arr = {}
    -- construct array of features
    ...
    local ml_result = ml_model.predict(features_arr)
    util.waf_debug(m, "ml_result", ml_result)
    return ml_result > 0
end

function main()
    util.waf_debug(m, "Starting script execution")
    local rules = m.getTriggeredRules()
    ...
    if waf_mode == 'LEARNING_UNKNOWN' or waf_mode == 'PRODUCTION' then
        if should_block(m, rules) then
            m.setvar("TX.WAF_REQUEST_BLOCKED", "1")
            return 1
        end
    end
end
```

```
m.setvar("TX.WAF_REQUEST_BLOCKED", "0")  
return nil  
end
```

Đến bước này, cả việc huấn luyện và tích hợp mô hình xem như đã hoàn thành. Trong chương kế tiếp, chúng tôi sẽ xây dựng một công cụ để kiểm thử cho các trường hợp tấn công sau khi đã tích hợp mô hình học máy vào hệ thống WAF.

Xây dựng công cụ kiểm thử

5.1 Mục tiêu

Trước khi bắt đầu nội dung của chương này, chúng tôi xin nói sơ qua về mục tiêu của công cụ kiểm thử sắp hiện thực. Công cụ này ban đầu chúng tôi định dùng để kiểm thử cho cả hai trường hợp, tức là cả tấn công và request bình thường. Tuy nhiên, sau một thời gian dài thử nghiệm, chúng tôi nhận thấy việc chuẩn bị các testcase để kiểm thử cho các trường hợp bình thường tỏ ra không hiệu quả bởi tính đa dạng về hành vi của các website trong thực tế. Do đó, công cụ này chỉ được chúng tôi sử dụng để kiểm thử tấn công trên ứng dụng DVWA đã dựng sẵn, các testcase hay còn gọi là các vector tấn công được chúng tôi thu thập từ nhiều nguồn khác nhau, nhưng chiếm đa số là từ <https://github.com/swisskyrepo/PayloadsAllTheThings>. Đây là một nguồn tổng hợp rất nhiều vector tấn công khác nhau, được xây dựng bởi cộng đồng nhiều chuyên gia bảo mật. Dựa trên những vector tấn công này, chúng tôi sẽ xây dựng nên công cụ nhằm kiểm thử cho WAF. Công cụ này là một phần bổ sung cho tập kiểm thử dùng trong mô hình học máy, chủ yếu khai thác các trường hợp thực tế trên ứng dụng DVWA. Do giới hạn môi trường kiểm thử tấn công nên công việc kiểm thử có phần nào khó khăn. Ngoài ra, công cụ này vẫn chỉ mới tập trung vào một số lỗi khá cơ bản và hầu như chưa khai thác hết nhiều loại hình tấn công mới đang xuất hiện ngày càng đa dạng trên các website thực tế. Chúng tôi đặt tên cho công cụ này là **waf-test**.

5.2 Giới thiệu một số công nghệ được sử dụng

5.2.1 Node.js

Node.js là một môi trường cho phép thực thi ngôn ngữ Javascript ¹ (Javascript runtime environment). Với ưu điểm đa nền tảng, mã nguồn mở, sử dụng ngôn ngữ JavaScript,

¹ Javascript: ngôn ngữ lập trình phổ biến nhất được dùng để thực thi mã nguồn (code) phía người dùng trong các trình duyệt

Node.js đang ngày càng trở nên thịnh hành trong việc lập trình ứng dụng phía server.

Các cuộc đua trình duyệt bắt đầu từ năm 1995 [16] đã sản sinh ra rất nhiều JS engine với hiệu năng tính toán - xử lý rất cao, chẳng hạn như

- V8 Engine dùng trong trình duyệt Chrome và Chromium của Google.
- SpiderMonkey dùng trong trình duyệt Firefox của Mozilla.
- JavaScriptCore dùng trong trình duyệt Safari của Apple.
- Chakra Engine dùng trong trình duyệt Internet Explorer và Microsoft Edge của Microsoft.
- Hermes Engine cho các ứng dụng Android dùng React Native của Facebook.

Với việc dùng V8 engine² để thực thi mã Javascript, Node.js hiện tại là một trong những nền tảng tốt nhất cho lập trình ứng dụng phía server cũng như các ứng dụng trên dòng lệnh (command line application). Cú pháp ngắn gọn, linh hoạt của ngôn ngữ Javascript cùng với tốc độ xử lý nhanh, nền tảng thư viện đồ sộ của Node.js là một trong những yếu tố quan trọng khiến cho chúng tôi chọn Node.js để hiện thực công cụ kiểm thử cho WAF.

5.2.2 Puppeteer

Puppeteer là một thư viện của Node.js cung cấp các API cấp cao dùng để điều khiển trình duyệt Chrome và Chromium thông qua DevTool protocol. Puppeteer có thể chạy ở hai chế độ là **headless** và **non-headless**

- Headless: Trong chế độ này, trình duyệt sẽ được khởi động mà không có giao diện người dùng. Thích hợp trong khi viết các công cụ kiểm thử tự động chạy ngầm (background).
- Non-headless: Đây là chế độ có hiển thị giao diện như trình duyệt thông thường. Dùng để điều khiển một trình duyệt đang chạy hoặc mở một trình duyệt mới. Chế độ này có thể dùng để kiểm tra công cụ (debug) trước khi chạy ở chế độ headless.

5.3 Chuẩn bị các testcase

Như đã đề cập trong phần mục tiêu, các testcase của chúng tôi được lấy từ nhiều nguồn và chủ yếu từ <https://github.com/swisskyrepo/PayloadsAllTheThings>. Do thời gian có hạn, các testcase của chúng tôi sẽ chỉ tập trung vào ba loại tấn công cơ bản là **SQL Injection**, **XSS** và **Command Injection**. Toàn bộ dữ liệu testcase mà chúng tôi sử dụng trong công cụ có thể tham khảo trong thư mục `tools/waf-test/data`. Dưới đây là một

²V8 engine: là một Javascript engine với hiệu năng cao, được viết bằng ngôn ngữ C++ và phát triển bởi Google

số testcase tấn công thuộc thể loại SQL Injection.

```
1 or 1=1
a'
a' something
aaaa' or true #
aaaa' union select 1 #
1' or 1=1 --
1" or 1=1 --
1' Or true
1' ||1
1' ||true
' or 1=1 limit 1 -- -
' or 'x'='x
-1'<@=1 or {a 1}=1 OR '
-1' AND 2<@ UNION/#!SELECT*/1, version()'
-1'<@=1 OR {x (select 1)}='1
1'/*comment*/or/**/1=1/**/--
a'+(SELECT 3)+'
```

5.4 Nguyên lý hoạt động của công cụ

Cách công cụ hoạt động cũng như tổng hợp kết quả khá đơn giản. Công cụ sẽ sử dụng các vector tấn công trong tập dữ liệu để sinh ra các request tấn công trên ứng dụng DVWA thông qua công cụ headless browser **Puppeteer**. Với mỗi request tấn công, nếu WAF phát hiện và ngăn chặn được thì được tính là một test case thành công và ngược lại. Toàn bộ kết quả trong quá trình kiểm thử sẽ được lưu lại dưới định dạng CSV, định dạng này giúp chúng ta có thể dễ dàng phân tích kết quả bằng các công cụ bảng tính như Excel hay Google Sheet.

5.4.1 Cấu trúc các module

Về cơ bản, công cụ này sẽ bao gồm ba module chính.

Chrome module. Đây là module được viết như một wrapper cho một tác vụ liên quan đến việc điều khiển trình duyệt, dựa trên các API mà thư viện Puppeteer cung cấp. Các tác vụ này ví dụ như mở trình duyệt, đi đến trang web nào đó bằng URL, xóa cookie, submit form,...

```
async function openBrowser() {...}

async function submit(inputs, submitSelector) {...}

async function isBlocked() {...}

async function clearSiteData() {...}
```

```
async function goTo(url) {...}

async function setCookie(cookies) { ... }
```

Utils module. Đây là module cho phép thực hiện một số tác vụ khác như đọc file, thêm vào file kết quả, xóa tất cả file trong thư mục,...

```
function removeFilesInFolder(directory) {...}

function* readTxtFile(fileName) {...}

function appendResult(url, payload, expect, result, options) {...}
```

DVWA module. Module này chứa các hàm liên quan trong phạm vi của ứng dụng DVWA. Các hàm này thực hiện một số thao tác như nhập vector tấn công, chỉnh mức độ bảo mật trong DVWA, đăng nhập, kiểm tra xem có bị chặn bởi WAF,...

```
async function test(options, manifestPath, outputPath) { ... }

async function reset() { ... }

async function setSecurityLevel(level = "low") { ... }

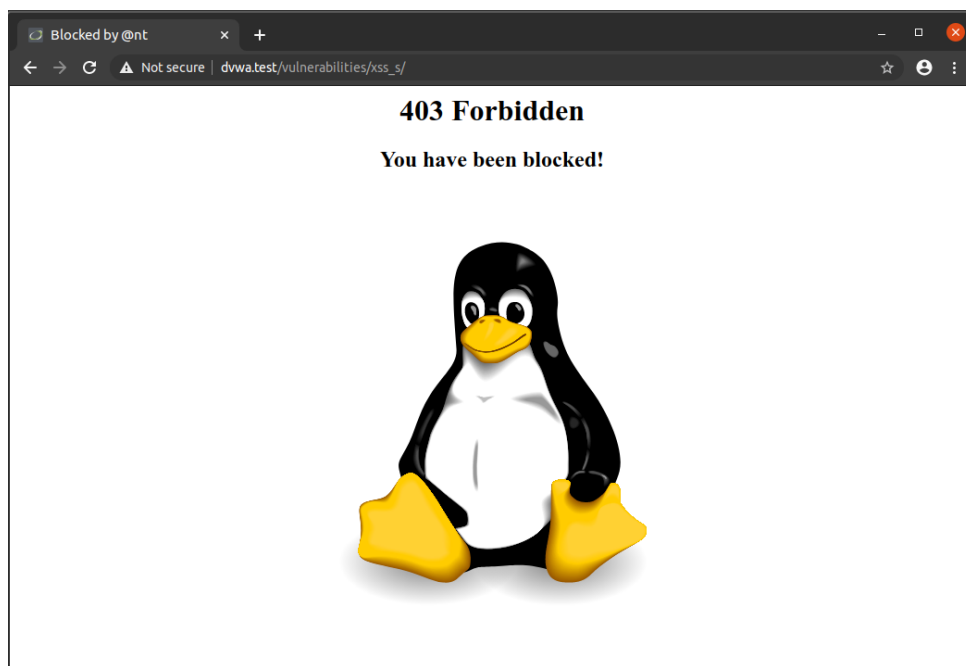
async function login(username = "admin", password = "password") { ... }

function isBlockedByFirewall(html) { ... }

async function testDVWA({config, ignoreDialog, verbose, headless, proxy
  }) { ... }
```

5.4.2 Dấu hiệu khi WAF phát hiện tấn công

Trong quá trình kiểm thử, làm thế nào để phát hiện liệu chúng ta có đang bị chặn bởi WAF hay không? Cách đơn giản là dựa vào nội dung trả về của ứng dụng. Trong hình 5.1 là một ví dụ sau khi WAF phát hiện tấn công từ người dùng.



Hình 5.1: Tấn công bị WAF phát hiện và ngăn chặn

Dựa vào thông báo 403 Forbidden hiện ra từ trang chủ của ứng dụng web, ta có thể hiện thực phần kiểm tra như sau

```
/*
Add this line in .env file

BLOCK_STRING='403 Forbidden'
*/

async function isBlocked() {
  let pageHtml = await page.evaluate(() => {
    return document.body.innerHTML;
  });

  return pageHtml.indexOf(process.env.BLOCK_STRING) >= 0;
}
```

5.4.3 Cấu trúc của test case trong DVWA module

Để thuận tiện cho việc bổ sung, quản lý các test case cũng như dễ dàng xử lý với ngôn ngữ Javascript, chúng tôi sử dụng định dạng JSON. Dưới đây là một cấu hình minh họa của các test case trong đường dẫn `manifest/dvwa.json` tính từ thư mục gốc chứa mã nguồn của công cụ.

[

```

{
  "url": "/vulnerabilities/xss_s/",
  "payloadFile": "../data/xss-attack-generic.txt",
  "type": "Xss generic",
  "testLocation": {
    "selector": "textarea[name=mtxMessage]",
    "submit": "input[type=submit]"
  },
  "constValues": [
    {
      "selector": "input[name=txtName]",
      "value": "something"
    }
  ]
},
...
]

```

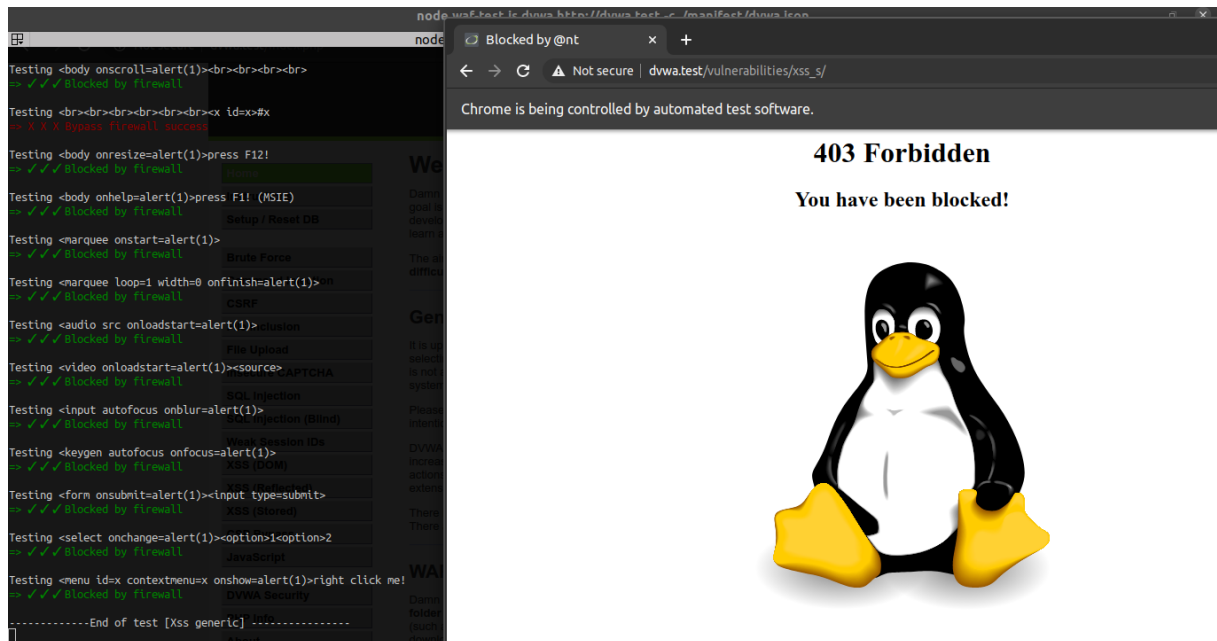
Trong đó, các tham số cấu hình có thể được giải thích ngắn gọn như sau

- **url**: đường dẫn tới url cần kiểm thử (host được cấu hình trong file .env)
- **payloadFile**: đường dẫn tới file chứa các vector tấn công
- **type**: tên của thể loại tấn công
- **testLocation**:
 - **selector**: css selector của input cần kiểm tra trong trang
 - **submit**: css selector của nút submit trong trang
- **constValues**:
 - **selector**: css selector của input chứa giá trị cố định
 - **value**: giá trị cố định cần submit

Theo cấu hình trên, công cụ này kiểm thử đối với trang web DVWA bằng cách đi vào từng endpoint được liệt kê, thử từng dòng trong bộ dữ liệu tấn công được chuẩn bị sẵn lên trường input được cấu hình cùng với các giá trị mặc định (const values). Công cụ sẽ sinh ra file kết quả sau khi hoàn thành kiểm thử. Công cụ khi chạy sẽ như trong hình 5.2.

5.4.4 Định dạng của file kết quả

Sau khi đã kiểm thử toàn bộ các test case được liệt kê trong file cấu hình tại đường dẫn `manifest/dvwa.json`, công cụ sẽ sinh các file kết quả ứng với từng kiểu test và đặt trong thư mục `outputs/`. Định dạng của file kết quả bao gồm 4 cột là URL chứa endpoint đến trang web đang test, Payload chứa attack vector, Expect chứa kết quả mong đợi nếu thành công, Result chứa kết quả của test case đó (đúng hoặc sai). Hình 5.3 là kết quả ví dụ sau khi kiểm thử với thể loại tấn công **SQL Injection**.



Hình 5.2: Chạy công cụ waf-test với ứng dụng DVWA

X

Sql injection-output.csv

File Edit Insert Format Help

Unsaved changes

↶ ↷ 🔍

Calibri

12

B
I
U
~~ABC~~

A

🗑️

📄

🔍

🔍

🔍

🔍

f^x

URL

	A	B	C	D	E	F	G	H
1	URL	Payload	Expect	Result				
2								
3	/vulnerabilities/sqli/	1 or 1=1	BLOCKED	SUCCESS				
4	/vulnerabilities/sqli/	a'	BLOCKED	FAILED				
5	/vulnerabilities/sqli/	a' something	BLOCKED	FAILED				
6	/vulnerabilities/sqli/	aaaa' or true #	BLOCKED	SUCCESS				
7	/vulnerabilities/sqli/	aaaa' union select 1 #	BLOCKED	SUCCESS				
8	/vulnerabilities/sqli/	1' or 1=1 --	BLOCKED	SUCCESS				
9	/vulnerabilities/sqli/	1" or 1=1 --	BLOCKED	SUCCESS				
10	/vulnerabilities/sqli/	1' Or true	BLOCKED	SUCCESS				
11	/vulnerabilities/sqli/	1' 1	BLOCKED	SUCCESS				
12	/vulnerabilities/sqli/	1' true	BLOCKED	SUCCESS				
13	/vulnerabilities/sqli/	' or 1=1 limit 1 --	BLOCKED	SUCCESS				
14	/vulnerabilities/sqli/	' or 'x'='x	BLOCKED	SUCCESS				
15	/vulnerabilities/sqli/	-1'<@=1 or {a 1}=1 OR '	BLOCKED	FAILED				
16	/vulnerabilities/sqli/	-1' AND 2<@ UNION/*!SEI	BLOCKED	SUCCESS				
17	/vulnerabilities/sqli/	-1'<@=1 OR {x (select 1)}=	BLOCKED	FAILED				
18	/vulnerabilities/sqli/	1'/*comment*/or/**/1=1/**/-	BLOCKED	SUCCESS				
19	/vulnerabilities/sqli/	a'+(SELECT 3)+'	BLOCKED	SUCCESS				

Hình 5.3: Kết quả kiểm thử với tấn công SQL Injection trên DVWA

Kiểm định và đánh giá kết quả

6.1 Kiểm định mô hình học máy

Với mô hình học máy được huấn luyện như đã trình bày trong Chương 4, chúng tôi thu được kết quả sau.

Mô hình Decision Tree

- Độ chính xác tập kiểm thử: 95,41%
- Điểm kiểm chứng chéo (cross validation): 94,79%

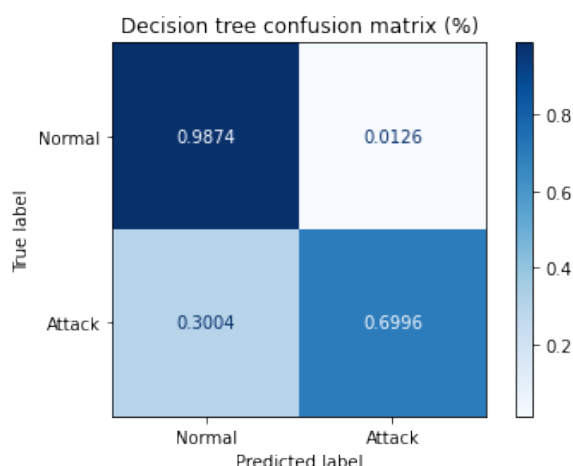
Mô hình Random Forest

- Độ chính xác tập kiểm thử: 95,43%
- Điểm kiểm chứng chéo (cross validation): 94,82%

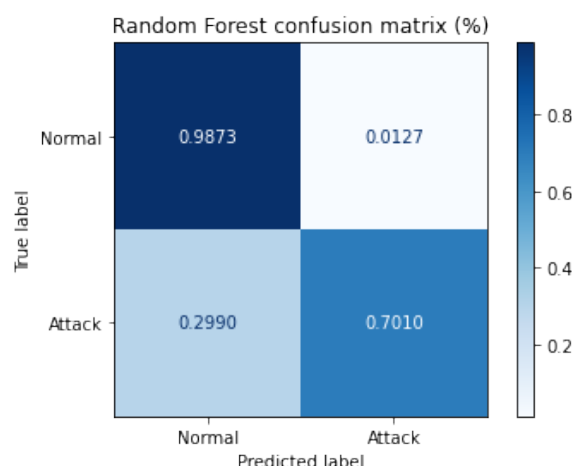
Lưu ý: điểm kiểm chứng chéo ở đây được thực hiện thông qua việc chia tập dữ liệu huấn luyện thành 5 phần theo phương pháp k-fold validation. Tập dữ liệu ban đầu được chia thành hai tập huấn luyện và kiểm thử với tỉ lệ 2:1 (xem lại Chương 4).

Nếu chỉ dựa vào kết quả trên, chúng ta dễ dàng lầm tưởng đây là một kết quả tốt. Tuy nhiên, nếu đánh giá kỹ hơn, ta sẽ thấy kết quả này chưa thực sự tốt. Trước hết chúng ta hãy xem hình 6.1 và 6.2 là kết quả của ma trận tương quan khi dự đoán trên tập kiểm thử với hai mô hình trên.

Dựa vào hình 6.1, có thể thấy rằng độ chính xác mà mô hình dự đoán đối với một request là tấn công chỉ ở mức 69,96%. Đối với một request bình thường, độ chính xác mà mô hình dự đoán được là 98,74%, con số này có vẻ khả quan hơn nhiều. Tuy nhiên, nếu nhìn ở góc độ ứng dụng, độ chính xác 98,74% là chưa đủ, thậm chí có thể nói là thấp. Chúng ta có thể lấy một ví dụ đơn giản, cứ với trung bình 100 request bình thường



Hình 6.1: Ma trận tương quan của mô hình Decision Tree



Hình 6.2: Ma trận tương quan của mô hình Random Forest

(chỉ tính những request khác nhau) đi vào hệ thống WAF ứng dụng mô hình này, sẽ có hơn 1 request nào đó bị hiểu nhầm là tấn công. Nếu hệ thống WAF trên được ứng dụng để bảo vệ một website khách hàng, chỉ cần một request bị chặn nhầm, có khi sẽ khiến người dùng không thể dùng được. Sẽ còn tệ hơn nếu người dùng bị chặn nhầm ngay bước mua hàng, hoặc đăng nhập vào hệ thống trên các website thương mại điện tử. Do đó, đặc thù của bài toán này là độ chính xác khi dự đoán lớp “bình thường” phải rất cao (trên 99,9%), trong khi độ chính xác dự đoán lớp “tấn công” chỉ cần ở mức chấp nhận được (khoảng trên 80%). Trong quá trình thử nghiệm mô hình học máy với nhiều thông số khác nhau, chúng tôi nhận ra việc cải thiện độ chính xác của mô hình trong trường hợp này là vô cùng khó khăn. Có thể nói, kết quả hiện tại là tương đối cân bằng, theo thử nghiệm thực tế, chỉ cần tăng thêm khoảng 1% độ dự đoán chính xác của lớp “bình thường” sẽ làm giảm hơn 10% (có thể hơn) độ dự đoán chính xác của lớp “tấn công”.

6.2 Kiểm định với công cụ waf-test

Với số lượng hạn chế của tập dữ liệu tấn công trong quá trình huấn luyện, chúng tôi đã sử dụng công cụ **waf-test** như một giải pháp bổ sung, nhằm kiểm tra độ chính xác của WAF khi dự đoán các request tấn công. Kết quả thu được sau khi sử dụng công cụ có thể tóm tắt trong bảng 6.1.

Dựa vào kết quả thực tế, chúng ta thấy rằng mô hình Random Forest tỏ ra hiệu quả hơn khá nhiều trong việc phát hiện tấn công so với mô hình Decision Tree. Ở đây, trong tập kiểm thử của chúng tôi bao gồm 300 testcase tấn công của công cụ **waf-test** (có thể xem lại Chương 5). Đối với mô hình Decision Tree, hệ thống WAF chặn được 124 trên tổng số 300 request tấn công. Đối với mô hình Random Forest, kết quả chặn được là 231

Bảng 6.1: Kết quả kiểm định với công cụ `waf-test`

Mô hình	Số lượng chặn được	Số lượng không chặn được
Decision Tree	124 (41,33%)	176 (58,67%)
Random Forest	231 (77%)	69 (23%)

trên tổng số 300. Mặc dù, các thể loại tấn công vẫn chưa đủ đa dạng nhưng kết quả thu được vẫn phần nào nói lên khả năng phát hiện tấn công của hệ thống WAF.

7.1 Các kết quả đạt được

Trong quá trình thực hiện đề tài “Ứng dụng học máy để cải thiện Web Application Firewall”, chúng tôi đã đề xuất một phương pháp tương đối hiệu quả trong việc sinh dữ liệu có nhãn nhằm phục vụ cho các mô hình học máy nói chung và cho đề tài này nói riêng.

Dựa trên tập dữ liệu sinh được cùng với dữ liệu có sẵn, chúng tôi đã đề xuất một phương pháp kết hợp các mô hình học máy Decision Tree và Random Forest với ModSecurity cùng tập luật ModSecurity CRS. Với mô hình đã huấn luyện, chúng tôi đã tích hợp thành công vào ModSecurity thông qua một số sửa đổi nhỏ trong thư viện. Kết quả là chúng tôi đã xây dựng được một hệ thống WAF tương đối hoàn chỉnh, cải thiện đáng kể độ *false positive* của tập luật ModSecurity CRS và qua đó có thể ứng dụng được cho các trường hợp thực tế.

Ngoài ra, chúng tôi cũng đã hiện thực công cụ kiểm thử tấn công `waf-test` nhằm bổ sung cho sự thiếu hụt về dữ liệu tấn công trong tập kiểm thử ban đầu.

7.2 Những hạn chế và hướng phát triển trong tương lai

Hiện tại, hệ thống WAF mà chúng tôi xây dựng đã có thể dùng được với một số website thực tế. Tuy nhiên, việc gặp những trường hợp *false positive* hay nói cách khác là chặn nhầm người dùng bình thường là không thể tránh khỏi. Khi gặp những trường hợp này trong quá trình sử dụng hằng ngày, hầu như chúng tôi phải tạm thời tắt hệ thống WAF. Do đó, việc phát triển một trang web quản lý, giúp người quản trị xem những tấn công hiện tại cũng như để đánh dấu các trường hợp *false positive* nhằm vá lỗi tạm thời là một việc cần thiết, nếu muốn đưa hệ thống vào áp dụng thực tế. Ngoài ra, có một

điểm cần lưu ý, đối với những request tấn công mà không kích hoạt bất kỳ luật nào trong ModSecurity CRS, việc dự đoán của mô hình sẽ không còn chính xác. Mô hình này, về cơ bản chỉ cải thiện lại công thức quyết định cuối cùng cho ModSecurity CRS chứ không cải thiện được khả năng phát hiện tấn công mới. Để cải thiện điều này, cách đơn giản là bổ sung thêm một tập luật mang tính tổng quát hơn và huấn luyện lại mô hình dựa trên tập luật đó.

Tài liệu tham khảo

- [1] Gustavo Betarte et al. “Machine learning-assisted virtual patching of web applications”. In: *arXiv preprint arXiv:1803.05529* (2018).
- [2] Gonzalo Álvarez Marañón Carmen Torrano Giménez Alejandro Pérez Villegas. *HTTP DATASET CSIC 2010*. <https://www.isi.csic.es/dataset/>. [Online; accessed 14-December-2019]. 2010.
- [3] Christian Folini Chaim Sanders Walter Hop. *OWASP ModSecurity Core Rule Set (CRS)*. <https://modsecurity.org/crs/>. [Online; accessed 14-December-2019]. 2006-2016.
- [4] CyberTrust. *Dataset Details*. https://www.impactcybertrust.org/dataset_view?idDataset=940. [Online; accessed 13-June-2020].
- [5] C. Folini. *Handling false positives with the owasp modsecurity core rule set*. https://www.netnea.com/cms/apache-tutorial-8_handling-false-positives-modsecurity-core-rule-set/. [Online; accessed 17-December-2019]. 2016.
- [6] Tammo Krueger et al. “TokDoc: A self-healing web application firewall”. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. ACM. 2010, pp. 1846–1853.
- [7] Abdelhamid Maklouf, Youcef Begriche, and Ahmed Serhrouchni. “Improving Web Application Firewalls to detect advanced SQL injection attacks”. In: *2014 10th International Conference on Information Assurance and Security*. IEEE. 2014, pp. 35–40.
- [8] Top OWASP. “Top 10-2017 The Ten Most Critical Web Application Security Risks”. In: *URL: owasp.org/images/7/72/OWASP_Top_10-2017_%20en* 29 (2017).
- [9] Ivan Ristić. *Modsecurity Handbook. The definitive guide to the popular open source web application firewall*. Feisty Duck, 2010.

- [10] Wei Rong, Bowen Zhang, and Xixiang Lv. “Malicious Web Request Detection Using Character-Level CNN”. In: *International Conference on Machine Learning for Cyber Security*. Springer. 2019, pp. 6–16.
- [11] Dewhurst Security. *Damn Vulnerable Web Application (DVWA)*. <http://www.dvwa.co.uk/>. [Online; accessed 13-December-2019]. 2019.
- [12] Trustwave’s SpiderLabs. *ModSecurity*. <https://github.com/SpiderLabs/ModSecurity/tree/v3/master>. [Online; accessed 13-December-2019]. 2019.
- [13] Trustwave’s SpiderLabs. *ModSecurity v3 Nginx Connector*. <https://github.com/SpiderLabs/ModSecurity-nginx>. [Online; accessed 15-December-2019]. 2004.
- [14] Trustwave’s SpiderLabs. *ModSecurity Wiki*. <https://github.com/SpiderLabs/ModSecurity/wiki>. [Online; accessed 12-December-2019]. 2018.
- [15] Camen Torrano-Giménez, Alejandro Perez-Villegas, and Gonzalo Alvarez Maranón. “An anomaly-based approach for intrusion detection in web traffic”. In: (2010).
- [16] Wikipedia. *Browser wars*. https://en.wikipedia.org/wiki/Browser_wars. [Online; accessed 17-December-2019]. 2019.
- [17] Ming Zhang et al. “A deep learning method to detect web attacks using a specially designed CNN”. In: *International Conference on Neural Information Processing*. Springer. 2017, pp. 828–836.